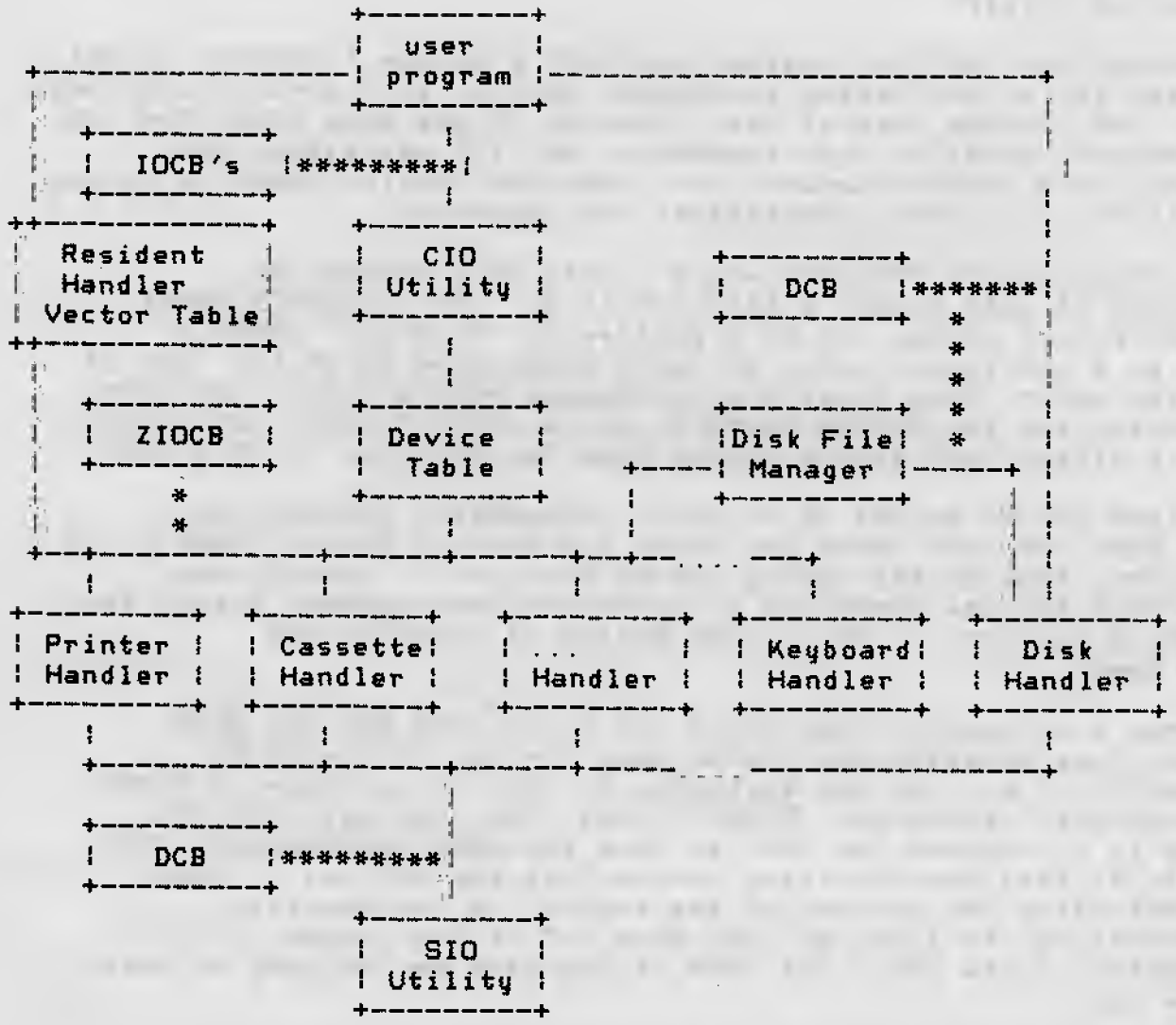
5 I/O SUBSYSTEM

This section discusses the I/O subsystem of the Operating System. The I/O subsystem comprises a collection of routines that allow you to access peripheral and local devices at three different levels. The CIO (Central I/O Utility), provides the highest level, device independent access to devices. The second level allows communication with the device handlers. The lowest level is the SIO (Serial I/O bus Utility) routine. Any lower level access to a device involves the direct reading and writing of the hardware registers associated with the device.

The data byte is the basic unit of input/output. A data byte can contain either "binary" (non text) information, or encoded text information. The text encoding scheme supported by the OS is called ATASCII, derived from the words "ATARI ASCII." Most ATASCII codes are the same as ASCII, with the primary deviations being the control codes. Appendix D shows the ATASCII character set, and Appendices E, F, and G show device-specific implementations for the display, keyboard, and printer.

The structure of the I/O subsystem is shown on the following page.



Where: ---- shows a control path. **** shows the data structure required for a path.

Note the following:

- The Keyboard/Display/Screen Editor Handlers don't use SIO.
- The Diskette handler cannot be called directly from CIO.
- The DCB is shown twice in the diagram.

Figure 5-1 I/O Subsystem Structure Flow Diagram

CENTRAL I/O UTILITY

The Central I/O Utility provides you with a single interface in which to access all of the system peripheral devices in a device-independent manner. The minimum unit of data transfer is the data byte. The CIO also supports multiple byte transfers. All I/O operations are performed on a "return-to-user-when-complete" basis; there is no way to initiate concurrent "overlapped" I/O processes.

I/O is organized by "files," where a file is a sequential collection of data bytes. A file can or may not contain textual data and it can or may not be organized by "records," where a record is a contiguous group of bytes terminated by an EOL (End of Line) character. Some files are synonymous with a device (as with the printer and the Screen Editor), while other devices can contain multiple files, each with a unique name (as with the disk drive).

CIO allows you to access up to eight independent device/files at one time, because there are eight I/O Control Blocks (IOCB's) in the system. Each of the IOCB's can be assigned to control any device/file because there are no preferred assignments, except that IOCB #0 is assigned to the Screen Editor at power-up and system reset.

To access a peripheral, you first set up an IOCB for the OPEN command, that supplies the system name for the device to be accessed (e.g. K:, for the keyboard, P:, for the printer, D:STARS for a diskette file named 'STARS', etc). You then call the CIO, telling it to examine the IOCB to find the OPEN information. CIO attempts to find the specified device/file and returns a status byte indicating the success of the search. If the specified device/file can be found by CIO, then CIO stores control information in the IOCB. The IOCB is now used for as long as that file is open.

Once a file is open, it can then be accessed using data-read or data-write types of commands; in general, reading can proceed until there is no more data to read (End of File) and writing can proceed until there is no more medium to store data on (End of Medium), although neither reading nor writing need proceed to that point. The reading and writing of data generally occurs into and out of user-supplied data buffers (although a special case allowing single byte transfers using the 6502 A register is provided).

When there are no more accesses to be performed on an open device/file, you perform the close operation. This accomplishes two functions:

- It terminates and makes permanent an output file (essential for diskette and cassette).

- It releases that IOCB to be used for another I/O operation.

CIO Design Philosophy

The CIO utility was designed specifically to meet the following design criteria.

- The transfer of data is device independent.
- Byte-at-a-time, multiple byte and record-aligned accesses are supported.
- Multiple device/files can be accessed concurrently.
- Error handling is largely device independent.
- New device handlers can be added without altering the system ROM.

Device Independence

CIO provides device independence by having a single entry point for all devices (and for all operations) and by having a device-independent calling sequence. Once a device/file is opened, data transfers occur with no regard to the actual device involved. Uniform rules for handling byte- and record-oriented data transfers allow the actual device storage block sizes to be transparent to you.

Data Access Methods

The CIO supports two file access methods: byte-aligned and record-aligned.

Byte-aligned accesses allow you to treat the device/file as a sequential byte stream; any number of bytes can be read or written and the following operation will continue where the prior one left off. Records are of no consequence in this mode, and reads or writes can encompass multiple records if desired.

Record-aligned accesses allow you to deal with the data stream at a higher level, that of the data record or "line of text." Each and every write operation creates a single record (by definition). Each read operation assures that the following read operation will start at the beginning of a record. Record-aligned accesses cannot deal with portions of more than one record at a time. Record-aligned accesses are useful only with text data or with binary data guaranteed not to contain the EOL character ($9B) as data.

Note that any file can be accessed using the byte-aligned access method, regardless of how the file was created. But not all files can be successfully read using record-aligned accesses; the file

must contain EOL characters at the end of each record and at no other place.

Multiple Device/File Concurrency

Up to eight device/files can be accessed concurrently using CIO, each operating independently of the others.

Unified Error Handling

All error detection and recovery occurs within the CIO subsystem. The status information that reaches you is in the form of a status byte for each device/file. Error codes are device independent as much as possible (see Appendix B).

Device Expansion

Devices are known by single character names such as K or P, and a number of device handlers are part of the resident system ROM. However, additional device handlers can be added to the system using the RAM-resident device table; this is normally done at power-up time as with the diskette boot process, but can be done at any point in time.

CIO Calling Mechanism

The input/output control block (IOCB) is the primary parameter passing structure between you and CIO. There are eight IOCB's in the system, arranged linearly in RAM as shown below:

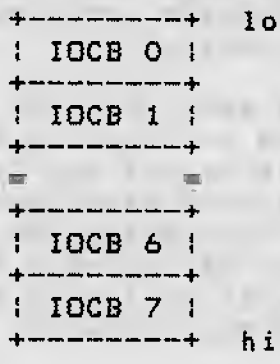


Figure 5-2 CIO Calling Mechanism

One IOCB is required for each open device/file. Any IOCB can be used to control any device/file, although IOCB 0 is normally assigned to the Screen Editor (E:). You perform a typical I/O operation by:

- Inserting appropriate parameters into an IOCB of your choosing
- Putting the IOCB number times 16 into the 6502 X register
- Performing a JSR to the CIO entry point CIOV [E456].

CIO returns to you when the operation is complete or if an error was encountered. The operation status is in the IOCB used, as well as in the 6502 Y register. The 6502 condition codes will also reflect the value in the Y register. In some cases a data byte will be in the 6502 A register. The X register will remain unchanged for all operations and conditions. An example is shown below:

```
IOCB2X = $20            ; INDEX FOR IOCB #2.

        LDX     #IOCB2X
        JSR     CIOV
        CPY     #0      ;(optional)
        BMI     ERROR
```

This sector describes each IOCB byte, with its file name and address. Each IOCB is 16 bytes long. Some bytes can be altered by you and some are reserved for use by CIO and/or the device handlers.

Handler ID -- ICHID [0340]

The handler ID is an index into the system device table (see Section 9) and is not user-alterable. This byte is set by CIO as the result of an OPEN command and is left unchanged until the device/file is closed, at that time CIO will set the byte to $FF.

Device Number -- ICDNO [0341]

The device number is provided by CIO as the result of an OPEN command and is not user-alterable. This byte is used to distinguish between multiple devices of the same type, such as D1: and D2:.

Command Byte -- ICCMD [0342]

You set the command byte. It specifies the command to be performed by the CIO. This byte is not altered by CIO.

Status -- ICSTA [0343]

The CIO conveys operation status to you with the command status byte as a result of each and every CIO call. Each and every CIO call updates the command status byte. The most significant (sign) bit is a one for error conditions and zero for non-error conditions, and the remaining bits represent an error number. See Appendix B for a list of status codes.

Buffer Address -- ICBAL [0344] and ICBAH [0345]

You set this 2-byte pointer; it is not altered by CIO. The pointer contains the address of the beginning (low address) of a buffer that:

- Contains data for read and write operations
- Contains the device/filename specification for the OPEN command.

You can alter the pointer at any time.

PUT Address -- ICPTL [0346] and ICPTH [0347]

The CIO sets this 2-byte pointer at OPEN time to the handler's PUT CHARACTER entry point (- 1). The pointer was provided to accommodate the people writing the ATARI BASIC cartridge, and has no legitimate use in the system. This variable is set to point to CIO's "IOCB not OPEN" routine on CLOSE, Power-up and [SYSTEM.RESET].

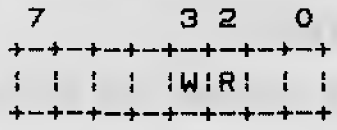
Buffer Length/Byte Count -- ICBLL [0348] and ICBLH [0349]

You set this 2-byte count to indicate the size of the data buffer pointed to by ICBAL and ICBAH for read and write operations. It is not required for OPEN. After each read or write operation, CIO will set this parameter to the number of bytes actually transferred into or out of the data buffer. For record-aligned access, the record length can well be less than the buffer length. Also an end of file condition or an error can cause the byte count to be less than the buffer length.

Auxiliary Information -- ICAX1 [034A] and ICAX2 [034B]

You set these 2-bytes. They contain information that is used by the OPEN command process and/or is device-dependent.

For OPEN, two bits of ICAX1 are always used to specify the OPEN direction as shown below, where R is set to 1 for input (read) enable and W is set to 1 for output (write) enable.



ICAX1 is not altered by CIO. You should not alter ICAX1 once the device/file is open.

The remaining bits of ICAX1 and all of ICAX2 contain only device-dependent data and are explained later in this section.

Remaining Bytes (ICAX3-ICAX6)

The handler reserves the four remaining bytes for processing the I/O command for CIO. There is no fixed use for these bytes. They are not user-alterable except as specified by the particular device descriptions. These bytes will be referred to as ICAX3, ICAX4, ICAX5 and ICAX6, although there are no equates for those names in the OS equate file.

CIO Functions

The CIO supports records and blocks and the handlers support single bytes. All of the system handlers support one or more of the eight basic functions subject to restrictions based upon the direction of data transfer (e.g. one cannot read data from the printer). The basic functions are: OPEN, CLOSE, GET CHARACTERS, PUT CHARACTERS, GET RECORD, PUT RECORD, GET STATUS, and SPECIAL.

OPEN -- Assign Device/Filename to IOCB and Ready for Access

A device/file must be opened before it can be accessed. This process links a specific IOCB to the appropriate device handler, initializes the device/file, initializes all CIO control variables, and passes device-specific options to the device handler.

You set up the following IOCB parameters prior to calling CIO for an OPEN operation:

COMMAND BYTE = $03

BUFFER ADDRESS = pointer to a device/filename specification.

AUX1 = OPEN direction bits, plus device-dependent information.

AUX2 = device-dependent information.

After an OPEN operation, CIO will have altered the following IOCB parameters:

HANDLER ID = index to the system device table; this is used only by CIO and must not be altered.

DEVICE NUMBER = device number taken from the device/filename specification and must not be altered.

STATUS = result of OPEN operation; see Appendix B for a list of the possible status codes. In general, a negative status will indicate a failure to open properly.

PUT ADDRESS = pointer to the PUT CHARACTERS routine for the device handler just opened.

It is recommended that this pointer not be used.

CLOSE -- Terminate Access to Device/File and Release IOCB.

You issue a CLOSE command after you are through accessing a given device/file. The CLOSE process completes any pending data writes, goes to the device handler for any device-specific actions, and then releases the IOCB.

You set the following IOCB parameter prior to calling CIO:

COMMAND BYTE = $0C

The CIO alters the following IOCB parameters as a result of the CLOSE operation:

HANDLER ID = $FF

STATUS = Result of CLOSE operation.

PUT ADDRESS = pointer to "IOCB not OPEN" routine.

GET CHARACTERS -- Read n Characters (Byte-Aligned Access)

The specified number of characters are read from the device/file to the user-supplied buffer. EOL characters have no termination features when using this function; there can be no EOL, or many EOL's, in the buffer after operation completion. There is a special case provided that passes a single byte of data in the 6502 A register when the buffer length is set to zero.

You set the following IOCB parameters prior to calling CIO:

COMMAND BYTE = $07

BUFFER ADDRESS = pointer to data buffer.

BUFFER LENGTH = number of bytes to read; if this is zero, the data will be returned in the 6502 A register only.

The CIO alters the following IOCB parameters as a result of the GET CHARACTERS operation:

STATUS = result of GET CHARACTERS operation.

BYTE COUNT/BUFFER LENGTH = number of bytes read to the buffer. The BYTE COUNT will always equal the BUFFER LENGTH except when an error or an end-of-file condition occurs.

PUT CHARACTERS -- Write n Characters (Byte-Aligned Access)

The specified number of characters are written from the user-supplied buffer to the device/file. EOL characters have no buffer terminating properties, although they have their standard meaning to the device/file receiving them; no EOL's are generated by CIO. There is a special case that allows a single character to be passed to CIO in the 6502 A register if the buffer length is zero.

You set the following IOCB parameters prior to initiating the PUT CHARACTERS operation:

COMMAND BYTE = $0B

BUFFER ADDRESS = pointer to data buffer.

BUFFER LENGTH = number of bytes of data in buffer.

The CIO alters the following IOCB parameter as a result of the PUT CHARACTERS operation:

STATUS = result of PUT CHARACTERS operation.

GET RECORD -- Read Up To n Characters (Record-Aligned Access)

Characters are read from the device/file to the user-supplied buffer until either the buffer is full or an EOL character is read and put into the buffer. If the buffer fills before an EOL is read, then the CIO continues reading characters from the device/file until an EOL is read,, and sets the status to indicate that a truncated record was read. No EOL will be put at the end of the buffer.

You set the following IOCB parameters prior to calling CIO:

COMMAND BYTE = $05

BUFFER ADDRESS = pointer to data buffer.

BUFFER LENGTH = maximum number of bytes to read (including the EOL character).

The CIO alters the following IOCB parameters as a result of the GET RECORD operation:

STATUS = result of GET RECORD operation.

BYTE COUNT/BUFFER LENGTH = number of bytes read to data buffer; this can be less than the maximum buffer length.

PUT RECORD -- Write Up To n Characters (Record-Aligned Access)

Characters are written from the user-supplied buffer to the device/file until either the buffer is empty or an EOL character is written. If the buffer is emptied without writing an EOL character to the device/file, then CIO will send an EOL after the last user-supplied character.

You set the following IOCB parameters prior to calling CIO:

COMMAND BYTE = $09

BUFFER ADDRESS = pointer to data buffer.

BUFFER LENGTH = maximum number of bytes in buffer.

The CIO alters the following IOCB parameter as a result of the PUT RECORD operation:

STATUS = result of PUT RECORD operation.

GET STATUS -- Return Device-Dependent Status Bytes

The device controller is sent a STATUS command, and the controller returns four bytes of status information that are stored in DVSTAT [02EA].

You set the following IOCB parameters prior to calling CIO:

COMMAND BYTE = $0D

BUFFER ADDRESS = pointer to a device/filename specification if the IOCB is not already OPEN; see the discussion of the implied OPEN option below.

After a GET STATUS operation, CIO will have altered the following parameters:

STATUS = result of GET STATUS operation; see Appendix B for a list of the possible status codes.

DVSTAT = the four-byte response from the device controller.

SPECIAL -- Special Function

Any command byte value greater than $0D is treated by CIO as a special case. Since CIO does not know what the function is, CIO transfers control to the device handler for complete processing of the operation.

The user sets the following IOCB parameters prior to calling CIO:

COMMAND BYTE > $0D

BUFFER ADDRESS = pointer to a device/filename specification if the IOCB is not already open; see the discussion of the implied OPEN option below.

Other IOCB bytes can be set up, depending upon the specific SPECIAL command being performed.

After a SPECIAL operation, CIO will have altered the following parameters:

STATUS = result of SPECIAL operation; see Appendix B for a list of the possible status codes.

Other bytes can be altered, depending upon the specific SPECIAL command.

Implied OPEN Option

The GET STATUS and SPECIAL commands are treated specially by CIO; they can use an already open IOCB to initiate the process or they can use an unopened IOCB. If the IOCB is unopened, then the buffer address must contain a pointer to a device/filename specification, just as for the OPEN command; CIO will then open that IOCB, perform the specified command and then close the IOCB again.

Device/Filename Specification

As part of the OPEN command, the IOCB buffer address parameter points to a device/filename specification, that is a string of ATASCII characters in the following format:

```
<specification> ::= <device>[<number>]:[<filename>]<eol>

<device> ::= C!D!E!K!P!R!S
<number> ::= 1!2!3!4!5!6!7!8
<filename> has device-dependent characteristics.
<eol> ::= $9B
```

The following devices are supported at this writing:

```
C = Cassette drive
D1 through D8 = Floppy diskette drives *
E = Screen Editor
K = Keyboard
P = 40-column printer
P2 = 80-column printer *
R1 through R4 = RS-232-C interfaces *
S = Screen display
```

Devices flagged by asterisks (*) are supported by nonresident handlers.

If <number> is not specified, it is assumed to be 1.

The following examples show valid device/filename specifications:

```
C:              Cassette
D2:BDAT         File "BDAT" on disk drive #2
D:HOLD          File "HOLD" on disk drive #1
K:              Keyboard
```

I/O Example

The example provided in this section illustrates a simple example of an I/O operation using the CIO routine.

```
; This code segment illustrates the simple example of reading
; text lines (records) from a diskette file named TESTER on disk
; drive #1. All symbols used are equated within the program
; although many of the symbols are in the OS equate file.

; The program performs the following steps:
;
;       1. Opens the file 'D1:TESTER' using IOCB #3.
;       2. Reads records until an error or EOF is reached.
;       3. Closes the file.

; I/O EQUATES

EOL=    $9B                     ; END OF LINE CHARACTER.
IOCB3=  $30                     ; IOCB #3 OFFSET (FROM IOCB #0).

ICHID=  $0340                   ; (HANDLER ID -- SET BY CIO).
ICDNO=  ICHID+1                 ; (DEVICE # -- SET BY CIO).
ICCOM=  ICDNO+1                 ; COMMAND BYTE.
ICSTA=  ICCOM+1                 ; STATUS BYTE -- SET BY CIO.
ICBAL=  ICSTA+1                 ; BUFFER ADDRESS (LOW).
ICBAH=  ICBAL+1                 ; BUFFER ADDRESS (HIGH).
ICPTL=  ICBAH+1
ICPTH=  ICPTL+1
ICBLL=  ICPTH+1                 ; BUFFER LENGTH (LOW).
ICBLH=  ICBLL+1                 ; BUFFER LENGTH (HIGH).
ICAX1=  ICBLH+1                 ; AUX 1.
ICAX2=  ICAX1+1                 ; AUX 2.

OPEN=   $03                     ; OPEN COMMAND.
GETREC= $05                     ; GET RECORD COMMAND.
CLOSE=  $0C                     ; CLOSE COMMAND.

OREAD=  $04                     ; OPEN DIRECTION = READ.
OWRIT=  $08                     ; OPEN DIRECTION = WRITE.

EOF=    $88                     ; END OF FILE STATUS VALUE.

CIOV=   $E456                   ; CIO ENTRY VECTOR ADDRESS.


;
; FIRST INITIALIZE THE IOCB FOR FILE "OPEN".
;


        LDX     #IOCB3          ; SETUP TO ACCESS IOCB #3.
```

```
        LDA     #OPEN           ; SETUP OPEN COMMAND.
        STA     ICCOM,X

        LDA     #NAME           ; SETUP BUFFER POINTER TO ...
        STA     ICBAL,X         ; ... POINT TO FILENAME.
        LDA     #NAME/256
        STA     ICBAH,X

        LDA     #OREAD          ; SETUP FOR OPEN READ.
        STA     ICAX1,X

        LDA     #0              ; CLEAR AUX 2.
        STA     ICAX2,X


;
; "OPEN" THE FILE.
;

        JSR     CIOV            ; PERFORM "OPEN" OPERATION.
        BPL     TP10            ; STATUS WAS POSITIVE -- OK.

        JMP     ERROR           ; NO -- "OPEN" PROBLEM.


;
; SETUP TO READ A RECORD.
;

TP10    LDA     #GETREC         ; SETUP "GET RECORD" COMMAND.
        STA     ICCOM,X

        LDA     #BUFF           ; SETUP DATA BUFFER POINTER.
        STA     ICBAL,X
        LDA     #BUFF/256
        STA     ICBAH,X


;
; READ RECORDS.
;


LOOP    LDA     #BUFFSZ         ; SETUP MAX RECORD SIZE ...
        STA     ICBLL,X         ; ... PRIOR TO EVERY READ.
        LDA     #BUFFSZ/256
        STA     ICBLH,X

        JSR     CIOV            ; READ A RECORD.
        BMI     TP20            ; MAY BE END OF FILE.

;
; A RECORD IS NOW IN THE DATA BUFFER "BUFF".  IT IS TERMINATED BY
```

```
;  AN EOL CHARACTER, AND THE RECORD LENGTH IS IN "ICBLL" and "ICBLH".
;  THIS EXAMPLE WILL DO NOTHING WITH THE RECORD JUST READ.
;
        JMP     LOOP                    ; READ NEXT RECORD.

;
;  NEGATIVE STATUS ON READ -- CHECK FOR END OF FILE.
;

TP20    CPY     #EOF                    ; END OF FILE STATUS?
        BNE     ERROR                   ; NO -- ERROR.

        LDA     #CLOSE                  ; YES -- CLOSE FILE.
        STA     ICCOM,X

        JSR     CIOV                    ; CLOSE THE FILE.

        JMP     *                       ; *** END OF PROGRAM ***

;
;  DATA REGION OF EXAMPLE PROGRAM
;

NAME    .BYTE   "D1:TESTER",EOL

BUFFSZ= 80                              ; 80 CHARACTER RECORD MAX
                                                (INCLUDES EOL).

BUFF=   *                               ; READ BUFFER.
*=      *+BUFFSZ
        .END
```

Figure 5-3 An I/O Example

Device-Specific Information

This section provides device-specific information regarding the device handlers that interface to CIO.

Keyboard Handler (K:)

The keyboard device is a read only device with a handler that supports the following CIO functions:

OPEN
CLOSE
GET CHARACTERS
GET RECORD
GET STATUS (null function)

The Keyboard Handler can produce the following error statuses:

$80 -- [BREAK] key abort.
$88 -- end-of-file (produced by pressing [CTRL] 3).

The Keyboard Handler is one of the resident handlers. It has a set of device vectors starting at location E420.

The keyboard can produce any of the 256 codes in the ATASCII character set (see Appendix F). Note that a few of the keyboard keys do not generate data at the Keyboard Handler level. These keys are described below:

[/|\] - The ATARI key toggles a flag that enables/disables the inversion of bit 7 of each data character read. The Screen Editor editing keys are exempted from such inversion, however.

CAPS - The [CAPS/LOWR] key provides three functions:

[SHIFT][CAPS/LOWR] -- Alpha caps lock.
[CNTRL][CAPS/LOWR] -- Alpha [CTRL] lock.
[CAPS/LOWR] -- Alpha unlock.

The system powers up and will system reset to the alpha caps lock option.

Some key combinations are ignored by the handler, such as [CTRL] 4 through [CTRL] 9, [CTRL] O, [CTRL] 1, [CTRL] /, and all key combinations in that the [SHIFT] and [CTRL] keys are depressed simultaneously.

The [CTRL] 3 key generates an EOL character and returns EOF status.

The [BREAK] key generates an EOL character and returns BREAK status.

CIO Function Descriptions

The device-specific characteristics of the standard CIO functions (described earlier in this section) are detailed below:

OPEN

The device name is K, and the handler ignores any device number and filename specification, if included.

There are no device-dependent option bits in AUX1 or AUX2.

CLOSE

No special handler actions.

GET CHARACTERS and GET RECORD

The handler returns the ATASCII key codes to CIO as they are entered, with no facility for editing.

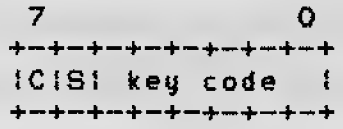
GET STATUS

The handler does nothing but set the status to $01.

Theory of Operation

Pressing a keyboard key generates an IRQ interrupt and vectors to the Keyboard Handler's interrupt service routine (see Section 6). The key code for the key pressed is then read and stored in data base variable CH [02FC]. This occurs whether or not there is an active read request to the Keyboard Handler, and effects a one-byte FIFO for keyboard entry. See Appendix L (E8) for a discussion of the auto repeat feature.

The Keyboard Handler monitors the CH variable for not containing the value $FF (empty state) whenever there is an active read request for the handler. When CH shows nonempty, the handler takes the key code from CH and sets CH to $FF again. The key code byte obtained from CH is not an ATASCII code and has the following form:



Where: C = 1 if the [CTRL] key is pressed.
S = 1 if the [SHIFT] key is pressed.

The remaining six bits are the hardware key code.

The key code obtained is then converted to ATASCII using the first of the following rules that applies:

1. Ignore the code if the C and S bits are both set.
2. If the C bit is set, process the key as a [CTRL] code.
3. If the S bit is set, process the key as a [SHIFT] code.
4. If [CTRL] lock is in effect, process alpha characters as CTRL codes, all others as lowercase.
5. IF [SHIFT] lock is in effect, process alpha characters as SHIFT codes, all others as lowercase.
6. Else, process as lowercase character.

Then: If the resultant code is not a Screen Editor control code, and if the video inverse flag is set, then set bit 7 of the ATASCII code (will cause inverse video when displayed).

KEY CODE TO ATASCII CONVERSION TABLE

Key Code	Key Cap	Lwr. Case	[SHIFT]	[CTRL]	Key Code	Key Cap	Lwr. Case	SHIFT	CTRL
00	L	6C	4C	0C	20	,	2C	5B	00
01	J	6A	4A	0A	21	SPACE	20	20	20
02	;	3B	3A	7B	22	.	2E	5D	60
03	--	--	--	--	23	N	6E	4E	0E
04	--	--	--	--	24	--	--	--	--
05	K	6B	4B	0B	25	M	6D	4D	0D
06	+	2B	5C	1E	26	/	2F	3F	--
07	*	2A	5E	1F	27	/¦\	--	--	--
08	O	6F	4F	0F	28	R	72	52	12
09	--	--	--	--	29	--	--	--	--
0A	P	70	50	10	2A	E	65	45	05
0B	U	75	55	15	2B	Y	79	59	19
0C	RET	9B	9B	9B	2C	TAB	7F	9F	9E
0D	I	69	49	09	2D	T	74	54	14
0E	-	2D	5F	1C	2E	W	77	57	17
0F	=	3D	7C	1D	2F	Q	71	51	11
10	V	76	56	16	30	9	39	28	--
11	--	--	--	--	31	--	--	--	--
12	C	63	43	03	32	0	30	29	--
13	--	--	--	--	33	7	37	27	--
14	--	--	--	--	34	BACKS	7E	9C	FE
15	B	62	42	02	35	8	38	40	--
16	X	78	58	18	36	<	3C	7D	7D
17	Z	7A	5A	1A	37	>	3E	9D	FF
18	4	34	24	--	38	F	66	46	06
19	--	--	--	--	39	H	68	48	08
1A	3	33	23	9B*	3A	D	64	44	04
1B	6	36	26	--	3B	--	--	--	--
1C	[ESC]	1B	1B	1B	3C	CAPS	--	--	--
1D	5	35	25	--	3D	G	67	47	07
1E	2	32	22	FD	3E	S	73	53	13
1F	1	31	21	--	3F	A	61	41	01

* [CTRL] 3 returns EOF status.

A complement of this table (ATASCII to keystroke) is given in Appendix F.

Figure 5-4 Keycode to ATASCII Conversion Table

Display Handler (S:)

The display device is a read/write device with a handler that supports the following CIO functions:

OPEN
CLOSE
GET CHARACTERS
GET RECORD
PUT CHARACTERS
PUT RECORD
GET STATUS (null function)
DRAW
FILL

The Display Handler can produce the following error statuses:

$84 -- Invalid special command.
$8D -- Cursor out-of-range.
$91 -- Screen mode > 11.
$93 -- Not enough memory for screen mode selected.

The Display Handler is one of the resident handlers, and therefore has a set of device vectors starting at location E410.

Screen Modes

You can operate the display screen in any of 20 configurations (modes 1 through 8, with or without split screen; plus mode 0, and modes 9 through 11 without split screen). Mode 0 is the text displaying mode. Modes 1 through 11 are all graphics modes (although modes 2 and 3 do display a subset of the ATASCII character set). Modes 9 through 11 require a GTIA chip to be installed in place of the standard CTIA chip.

TEXT MODE 0

In text mode 0 the screen is comprised of 24 lines of 40 characters per line. Program alterable left and right margins limit the display area. They default to 2 and 39 (of a possible 0 and 39).

A program-controllable cursor shows the destination of the next character to be output onto the screen. The cursor is visible as the inverse video representation of the current character at the destination position.

The text screen data is internally organized as variable length logical lines. The internal representation is 24 lines when the screen is cleared. Each EOL marks the end of a logical line as text is sent to the screen. If more than 3 physical lines of text are sent, a logical line will be formed every 3 physical lines. The number of physical lines used to comprise a logical line (1 to 3) is always the minimum required to hold the data for that logical line.

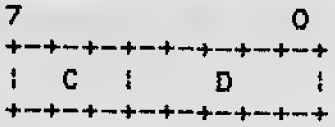
The text screen "scrolls" upward whenever a text line at the bottom row of the screen extends past the right margin, or a text line at the bottom row is terminated by an EOL. Scrolling removes the entire logical line that starts at the top of the screen, and then moves all subsequent lines upward to fill in the void. The cursor also moves upward, if the logical line deleted exceeds one physical line.

All data going to or coming from the text screen is represented in 8-bit ATASCII code as shown in Appendix E.

TEXT MODES 1 AND 2

In text modes 1 and 2 the screen comprises either 24 lines of 20 characters (mode 1), or 12 lines of 20 characters (mode 2). The left and right margins are of no consequence in these modes and there is no visible cursor. There are no logical lines associated with the data and in all regards these modes are treated as graphics modes by the handler.

Data going to or coming from the screen is in the form shown below:



Where: C is the color/character-set select field

C Value	Color (default)	Color Register (see Appendix H)	Character Set CHBAS=$E0	Character Set CHBAS=$E2
0	green	(PF1)	! – ?	[HEART] [ARROW]
1	gold	(PF0)	! – ?	[HEART] [ARROW]
2	gold	(PF0)	@ – _	[DIAMOND][TRIANGLE]
3	green	(PF1)	@ – _	[DIAMOND][TRIANGLE]
4	red	(PF3)	! – ?	[HEART] [ARROW]
5	blue	(PF2)	! – ?	[HEART] [ARROW]
6	blue	(PF2)	@ – _	[DIAMOND][TRIANGLE]
7	red	(PF3)	@ – _	[DIAMOND][TRIANGLE]

D is a 5-bit truncated ATASCII code that selects the specific character within the set selected by the C field. See Appendix E for the graphics representations of the characters.

Data base variable CHBAS [02F4] allows for the selection of either of two data sets. The default value of $E0 provides the capital letters, numbers and punctuation characters; the alternate value of $E2 provides lowercase letters and the special character graphics set.

Figure 5-5 Text Modes 1 and 2 Data Form

GRAPHICS MODES (Modes 3 Through 11)

The screen has varying physical characteristics for each of the graphics modes as shown in Appendix H. Depending upon the mode, a 1 to 16 color selection is available for each pixel and the screen size varies from 20 by 12 (lowest resolution) to 320 by 192 (highest resolution) pixels.

There is no visible cursor for the graphics mode output.

Data going to or coming from the graphics screen is represented as 1 to 8-bit codes as shown in Appendix H and in the GET/PUT diagrams following.

SPLIT-SCREEN CONFIGURATIONS

In split-screen configurations, the bottom of the screen is reserved for four lines of mode 0 text. The text region is controlled by the Screen Editor, and the graphics region is controlled by the Display handler. Two cursors are maintained in this configuration so that the screen segments can be managed independently.

To operate in split-screen mode, the Screen Editor must first be opened and then the Display Handler must be opened using a separate IOCB (with the split-screen option bit set in AUX1).

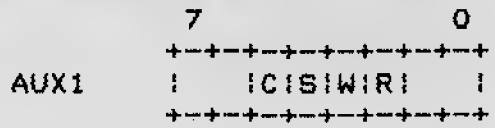
CIO Function Descriptions

The device-specific characteristics of the standard CIO functions (described earlier in this section) are detailed below:

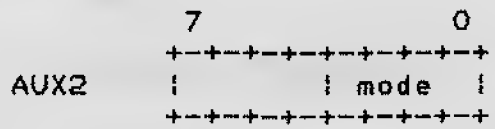
OPEN

The device name is S, and the handler ignores any device number and filename specification, if included.

The handler supports the following options:



Where: C = 1 indicates to inhibit screen clear on OPEN.
S = 1 indicates to set up a split-screen configuration (for modes 1 through 8 only).
R and W are the direction bits (read and write).



Where: mode is the screen mode (0 through 11).

Note: If the screen mode selected is 0, then the AUX1 C and S options are assumed to be 0.

You share memory utilization with the Display Handler information. Sharing is necessary because the Display Handler dynamically allocates high address memory for use in generating the screen display, and because different amounts of memory are needed for the different screen modes. Prior to initiating an OPEN command the variable APPMHI [000E] should contain the highest address of RAM you need. The Screen handler will open the screen only if no RAM is needed at or below that address.

Upon return from a screen OPEN, the variable MEMTOP [02E5] will contain the address of the last free byte at the end of RAM memory prior to the screen-required memory.

As a result of every OPEN command, the following screen variables are altered:

The text cursor is enabled (CRSINH = 0). The tabs are set to the default settings (2 and 39). The color registers are set to the default values (shown in Appendix H).

Tabs are set at positions 7,15,23,31,39, 47,55,63,71,79,87,95,103,111,119.

CLOSE

No special handler actions.

GET CHARACTERS and GET RECORD

Returns data in the following screen mode dependent forms, where each byte contains the data for one cursor position (pixel); there is no facility for having the handler return packed graphics data.

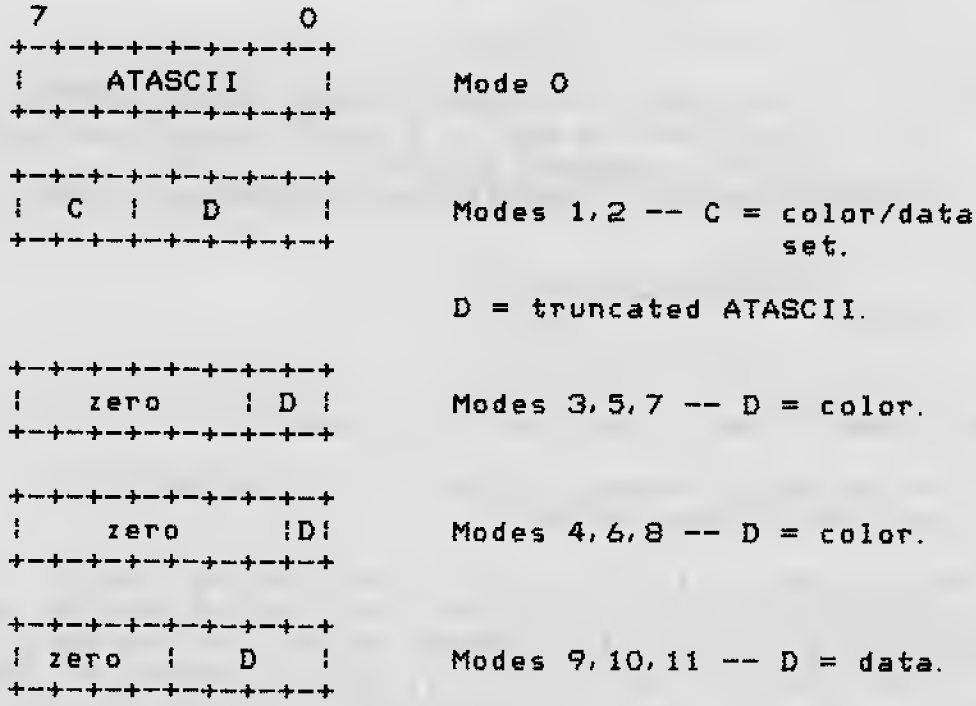


Figure 5-6 Graphics Mode 3-11 GET Data Form

The cursor moves to the next position as each data byte is returned. For mode 0, the cursor will stay within the specified margins; for all other modes, the cursor ignores the margins.

PUT CHARACTERS and PUT RECORD

The handler accepts display data in the following screen mode dependent forms; there is no facility for the handler to receive graphics data in packed form.

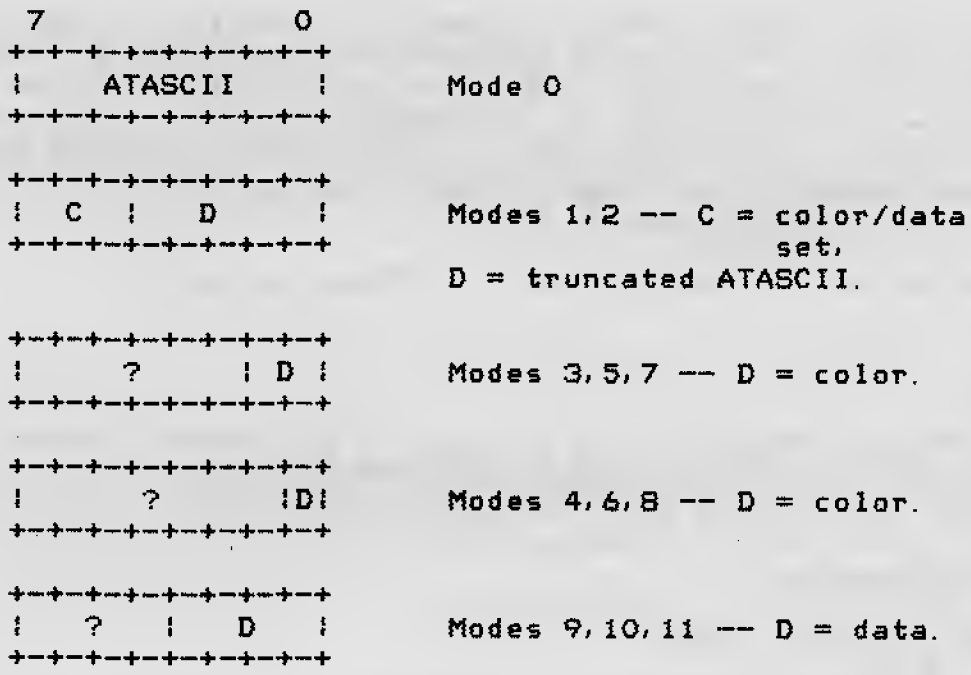


Figure 5-7 Graphics Mode 3-11 PUT Data Form

NOTE: For all modes, if the output data byte equals $9B (EOL), that byte will be treated as an EOL character; and if the output data byte equals $7D (CLEAR) that byte will be treated as a screen-clear character.

The cursor moves to the next cursor position as each data byte is written. For mode 0, the cursor will stay within the specified margins; for all other modes, the cursor ignores the margins.

While outputting, the Display Handler monitors the keyboard to detect the pressing of the [CTRL] 1 key combination. When this occurs, the handler loops internally until that key combination is pressed again: This effects a stop/start function that freezes the screen display. Note that there is no ATASCII code associated with either the [CTRL] 1 key combination or the start/stop function. The stop/start function can be controlled only from the keyboard (or by altering database variable CH as discussed in Appendix L, E4).

GET STATUS

No handler action except to set the status to $01.

DRAW

This special command draws a simulated "straight" line from the current cursor position to the location specified in ROWCRS [0054] and COLCRS [0055]. The color of the line is taken from the last character processed by the Display Handler or Screen Editor. To force the color, store the desired value in ATACHR [02FB]. At the completion of the command, the cursor will be at the location specified by ROWCRS and COLCRS.

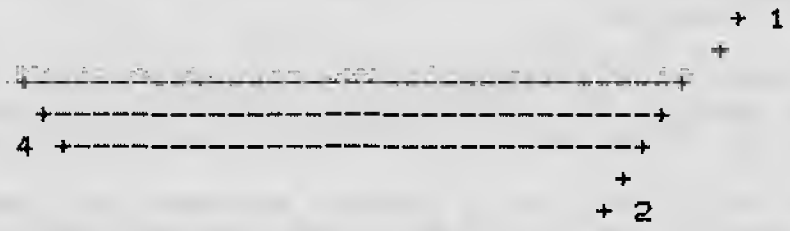
The value for the command byte for DRAW is $11.

FILL

This special command fills an area of the screen defined by two lines with a specified color. The command is set up the same as in DRAW, but as each point of the line is drawn, the routine scans to the right performing the procedure shown below (in PASCAL notation):

```
WHILE PIXEL [ROW,COL] = 0 DO
   BEGIN
      PIXEL [ROW,COL] := FILDAT;
      COL := COL + 1;
      IF COL > Screen right edge THEN COL := 0
   END;
```

An example of a FILL operation is shown below:



Where: '-' represents the fill operation.
'+' are the line points, with '+' for the endpoints.

1 -- set cursor and plot point.
2 -- set cursor and DRAW line.
3 -- set cursor and plot point.
4 -- set fill data value, set cursor, and FILL.

FILDAT [02FD] contains the fill data, and ROWCRS and COLCRS contain the cursor coordinates of the line endpoint. The value in ATACHR [02FB] will be used to draw the line; ATACHR always contains the last data read or written, so if the steps above are followed exactly, ATACHR will not have to be modified.

The value for the command byte for FILL is $12.

User-Alterable Data Base Variables

Certain functions of the Display Handler require you to examine and/or alter variables in the OS database. The following describes some of the more commonly used handler variables. (see Appendix L, B1-55, for additional descriptions).

Cursor Position

Two variables maintain the cursor position for the graphics screen or mode 0 text screen. ROWCRS [0054] maintains the display row number; and COLCRS [0055] maintains the display column number. Both numbers range from 0 to the maximum number of rows/columns, - 1. The cursor can be set outside of the defined text margins with no ill effect. You can read and write this region. The home position (0,0) for both text and graphics is the upper left corner of the screen.

ROWCRS is a single byte. COLCRS is maintained at 2-bytes, with the least significant byte being at the lower address.

When you alter these variables, the screen representation of the cursor will not move until the next I/O operation involving the display is performed.

Inhibit/Enable Visible Cursor Display

You can inhibit the display of the text cursor on the screen by setting the variable CRSINH [02F0] to any nonzero value. Subsequent I/O will not generate a visible cursor.

You can enable the display of the text cursor by setting CRSINH to zero. Subsequent I/O will then generate a visible cursor.

Text Margins

The text screen has user-alterable left and right margins. The OS sets these margins to 2 and 39. The variable LMARGN [0052] defines the left margin, and the variable RMARGN [0053] defines the right margin. The leftmost margin value is 0 and the

rightmost margin value is 39.

The margin values inclusively define the useable portion of the screen for all operations in that you do not explicitly alter the cursor location variables as described prior to this paragraph.

Color Control

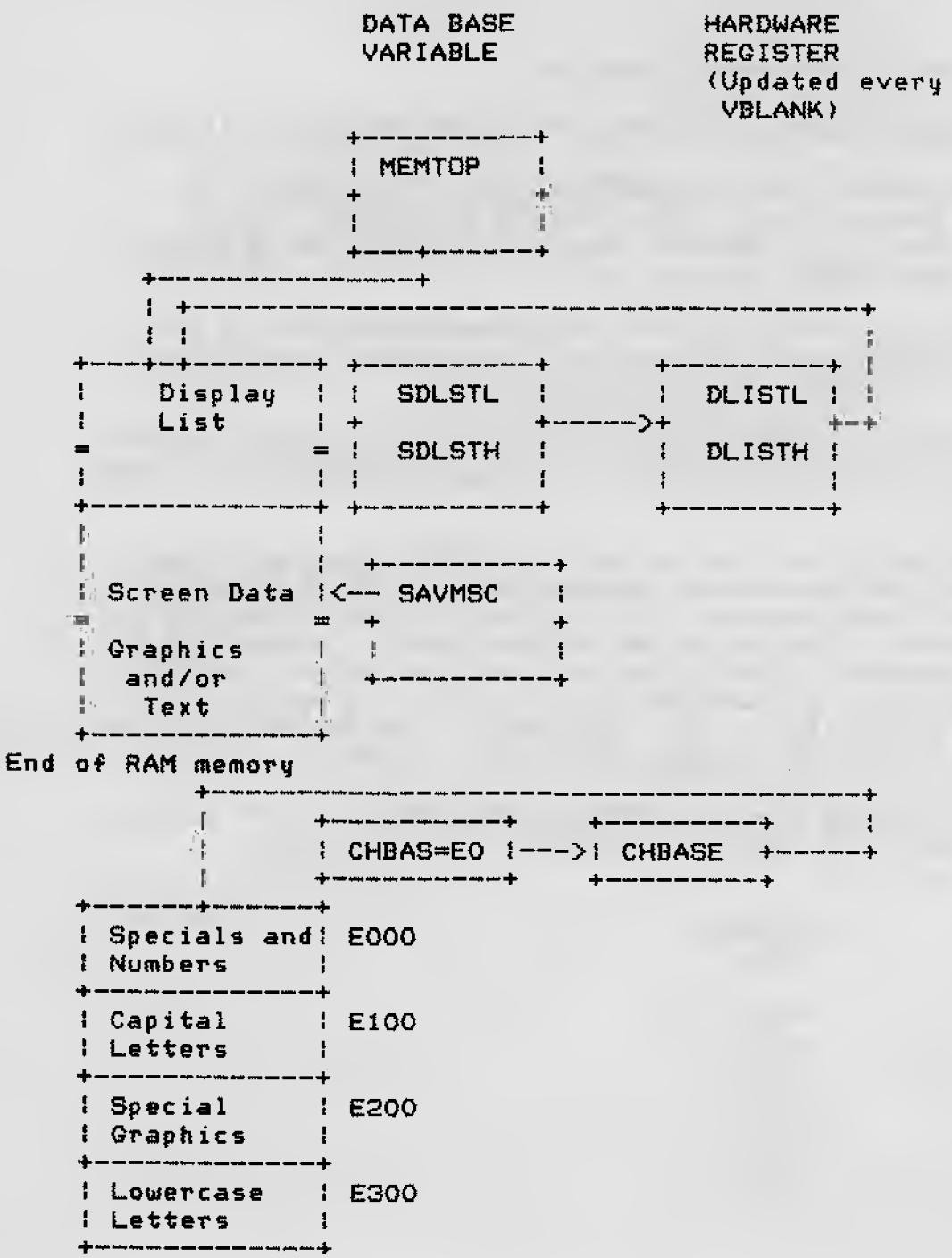
The OS updates hardware color registers using data from the OS data base as part of normal Stage 2 VBLANK processing (see Section 6). Shown below are the data base variable names, the hardware register names, and the function of each register. See Appendix H for the mode dependent uses for the registers.

Data Base	Hardware	Function
COLOR0	COLPF0	PF0 -- Playfield 0.
COLOR1	COLPF1	PF1 -- Playfield 1.
COLOR2	COLPF2	PF2 -- Playfield 2.
COLOR3	COLPF3	PF3 -- Playfield 3.
COLOR4	COLBK	BAK -- Playfield background.
PCOLR0	COLPM0	PM0 -- Player/missile 0.
PCOLR1	COLPM1	PM1 -- Player/missile 1.
PCOLR2	COLPM2	PM2 -- Player/missile 2.
PCOLR3	COLPM3	PM3 -- Player/missile 3.

Theory of Operation

The Display Handler automatically sets up all memory resources required to create and maintain the screen display at OPEN time. The screen generation hardware requires that two distinct data areas exist for graphics modes: 1) a display list and 2) a screen data region. A third data area must exist for text modes. This data area defines the screen representation for each of the text characters. Consult the ATARI Home Computer Hardware Manual for a complete understanding of the material that is to follow.

The simplified block diagram below shows the relationships between the memory and hardware registers used to set up a screen display (without player/missile graphics) by the OS Note that the hardware registers allow for many other possibilities.



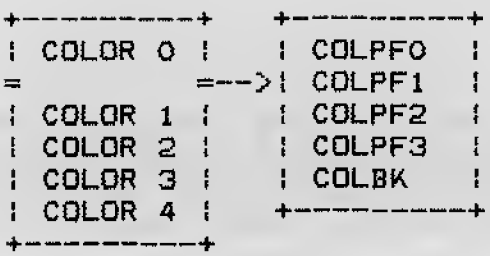


Figure 5-8 Screen Display Block Diagram

The following relationships are present in the preceding diagram:

1. Data base variables SDLSTL/SDLSTH contain the address of the current display list. This address is stored in the hardware display list address registers DLISTL and DLISTH as part of the VBLANK process.

2. The display list itself defines the characteristics of the screen to be displayed and points to the memory containing the data to be displayed.

3. Data base variable CHBAS contains the MSB of the base address of the character representations for the character data (text modes only).

   The default value for this variable is $E0. This variable declares that the character representations start at memory address E000 (the character set provided by the OS in ROM). Each character is defined as an 8X8 bit matrix, requiring 8 bytes per character. 1024 bytes are required to define the largest set, since a character code contains up to 7 significant bits (set of 128 characters). The OS ROM contains the default set in the region from E000 to E3FF.

   All character codes are converted by the handler from ATASCII to an internal code (and vice versa), as shown below:

ATASCII CODE	INTERNAL CODE
00-1F	40-5F
20-3F	00-1F
40-5F	20-3F
60-7F	60-7F
80-9F	C0-DF
A0-BF	80-9F
C0-DF	A0-BF
E0-FF	E0-FF

The character set in ROM is ordered by internal code order. Three considerations differentiate the internal code from the external (ATASCII) code:

ATASCII codes for all but the special graphics characters were to be similar to ASCII. The alphabetic, numeric, and punctuation character codes are identical to ASCII.

In text modes 1 and 2 it was desired that one character subset include capital letters, numbers, and punctuation and the other character subset include lowercase letters and special graphics characters.

The codes for the capital and lowercase letters were to be identical in text modes 1 and 2.

Database variables COLOR0 through COLOR4 contain the current color register assignments. Hardware color registers receive these values as part of the stage 1 VBLANK process, thus providing synchronized color changes (see Appendix H).

Database variable SAVMSC points to the lowest memory address of the screen data region. It corresponds to the data displayed at the upper left corner of the display.

When the Display Handler receives an open command, it first determines the screen mode from the OPEN IOCB. Then it allocates memory from the end of RAM downward (as specified by data base variable RAMTOP), first for the screen data and then for the display list. The screen data region is cleared and the display list is created if sufficient memory is available. The display list address is stored to the database.

Screen Editor (E:)

The Screen Editor is a read/write handler that uses the Keyboard Handler and the Display Handler to provide "line-at-a-time" input with interactive editing functions, as well as formatted output.

The Screen Editor supports the following CIO functions:

- OPEN
- CLOSE
- GET CHARACTERS
- GET RECORD
- PUT CHARACTERS
- PUT RECORD
- GET STATUS (null function)

See Keyboard Handler and Display Handler Sections for a discussion of Screen Editor error statuses.

The Screen Editor is one of the resident handlers, and therefore has a set of device vectors starting at location E400.

The Screen Editor is a program that reads key data from the Keyboard Handler and sends each character to the Display Handler for immediate display. The Screen Editor also accepts data from you to send to the Display Handler, and reads data from the Display Handler (not the Keyboard Handler) for you. In fact, the Keyboard Handler, Display Handler, and the Screen Editor are all contained in one monolithic hunk of code.

Most of the behaviors already defined for the Keyboard Handler and the Display Handler apply as well to the Screen Editor: The discussions in this Section will be limited to deviations from those behaviors, or to additional features that are part of the Screen Editor only. The Screen Editor deals only with text data (screen mode 0). This Section also explains the split-screen configuration feature.

The Screen Editor uses the Display Handler to read data from graphics and text screens on demand. You use the Screen Editor to determine when the program will read Screen data, and where upon the screen the data will be read from. You first locates the cursor on the screen to determine the screen area to be read; you then press the [RETURN] key to determine when the program will begin to read the data indicated.

When the [RETURN] key is pressed, the entire logical line within that the cursor resides is then made available to the calling program. Trailing blanks in a logical line are never returned as data, however. After all of the data in the line has been sent to the caller (this can entail multiple READ CHARACTERS functions if desired), an EOL character is returned and the cursor is positioned to the beginning of the logical line following the one just read.

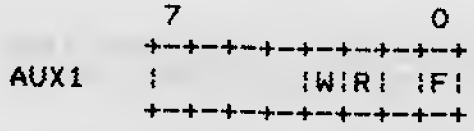
CIO Function Descriptions

The device-specific characteristics of the standard CIO functions are detailed below:

OPEN

The device name is E, and the Screen Editor ignores any device number and filename specification, if included.

The Screen Editor supports the following option:



Where: R and W are the direction bits (read and write).
F = 1 indicates that a "forced read" is desired (see GET CHARACTER and GET RECORD for more information).

CLOSE

No special handler actions.

GET CHARACTER and GET RECORD

Normally the Screen Editor will return data only when you press the [RETURN] key at the keyboard. However, the "forced read" OPEN option allows you to read text data without intervention. When you command a READ operation, the Screen Editor will return data from the start of the logical line in which the text cursor is located, and then move the cursor to the beginning of the following logical line. A read of the last logical line on the screen will cause the screen data to scroll.

A special case occurs when characters are output without a terminating EOL, and then additional characters are appended to

that logical line from the keyboard. When the [RETURN] key is pressed, only the keyboard entered characters are sent to the caller, unless the cursor has been moved out of and then back into the logical line, in that case all of the logical line will be sent.

PUT CHARACTER and PUT RECORD

The Handler accepts ATASCII characters as one character per byte. Sixteen of the 256 ATASCII characters are control codes; the EOL code has universal meaning, but most of the other control codes have special meaning only to a display or print device. The Screen Editor processing of the ATASCII control codes is explained below:

CLEAR ($7D) -- The Screen Editor clears the current display of all data and the cursor is placed at the home position (upper left corner of the screen).

CURSOR UP ($1C) -- The cursor moves up by one physical line. The cursor will wrap from the top line of the display to the bottom line.

CURSOR DOWN ($1D) -- The cursor moves down by one physical line. The cursor will wrap from the bottom line of the display to the top line.

CURSOR LEFT ($1E) -- The cursor moves left by one column. The cursor will wrap from the left margin of a line to the right margin of the same line.

CURSOR RIGHT ($1F) -- The cursor moves right by one column. The cursor will wrap from the right margin of a line to the left margin of the same line.

BACKSPACE ($7E) -- The cursor moves left by one column (but never past the beginning of a logical line), and the character at that new position is changed to a blank ($20).

SET TAB ($9F) -- The Screen Editor establishes a tab point at the logical line position at that the cursor is residing. The logical line tab position is not synonymous with the physical line column position since the logical line can be up to 3 physical lines in length. For example, tabs can be set at the 15th, 30th, 45th, 60th and 75th character positions of a logical line as shown below:

```
0 2       9         19        29        39  Screen column #.
--L-------+---------+---------+---------R   L/R = margins.

xx--------------T--------------T--------    A logical line.
xx------T--------------T--------------T-    x = inaccesible
xx-------------------------------------        columns.
```

Note the effect of the left margin in defining the limits of the logical line.

The Handler default tab settings are shown below:

```
0 2       9         19        29        39  Screen column #.
--L-------+---------+---------+---------R   L/R = margins.

xxT----T-------T-------T-------T-------T    A logical line.
xx-----T-------T-------T-------T-------T    x = inaccesible
xx-----T-------T-------T-------T-------T        columns.
```

CLEAR TAB ($9E) -- The Screen Editor clears the current cursor position within the logical line from being a tab point. There is no "clear all tab points" facility provided by the Handler.

TAB ($7F) -- The cursor moves to the next tab point in the current logical line, or to the beginning of the next line if no tab point is found. This function will not increase the logical line length to accommodate a tab point outside the current length (e.g. the logical line length is 38 characters and there is a tab point at position 50).

INSERT LINE ($9D) -- All physical lines at and below the physical line in that the cursor resides, are moved down by one physical line. The last logical line on the display can be truncated as a result. The blank physical line at the insert point becomes the beginning of a new logical line. A logical line can be split into two logical lines by this process, the last half of the original logical line being concatenated with the blank physical line formed at the insert point.

DELETE LINE ($9C) -- The logical line in that the cursor resides is deleted and all data below that line is moved upward to fill the void. Empty logical lines are created at the bottom of the display.

INSERT CHARACTER ($FF) -- All physical characters at and behind the cursor position on a logical line are moved one position to the right. The character at the cursor position is set to blank. The last character of the logical line will be lost when the logical line is full and a character is inserted. The number of physical lines comprising a logical line can increase as a result of this function.

DELETE CHARACTER ($FE) -- The character on which the cursor resides is removed, and the remainder of the logical line to the right of the deleted character is moved to the left by one position. The number of physical lines composing a logical line can decrease as a result of this function.

ESCAPE ($1B) -- The next non-EOL character following this code is displayed as data, even if it would normally be treated as a control code. The sequence [ESC][ESC] will cause the second [ESC] character to be displayed.

BELL ($FD) -- An audible tone is generated; the display is not modified.

END OF LINE ($9B) -- In addition to its record termination function, the EOL causes the cursor to advance to the beginning of the next logical line. When the cursor reaches the bottom line of the screen, the receipt of an EOL will cause the screen data to scroll upward by one logical line.

GET STATUS

The Handler takes no action other than to set the status to $01.

User-Alterable Data Base Variables

Also see the Display Handler data base variable discussion.

Cursor Position

When in a split-screen configuration, ROWCRS and COLCRS are associated with the graphics portion of the display and two other variables, TXTROW [0290] and TXTCOL [0291], are associated with the text window. TXTROW is a single byte, and TXTCOL is 2-bytes with the least significant byte being at the lower address. Note that the most significant byte of TXTCOL should always be zero.

The home position (0,0) for the text window is the upper left corner of the window.

Enable/Inhibit of Control Codes in Text

Normally all text mode control codes are operated upon as received, but sometimes it is desirable to have the control codes displayed as if they were data characters. This is done by setting the variable DSPFLG [02FE] to any nonzero value before outputting the data containing control codes. Setting DSPFLG to zero restores normal processing of text control codes.

Cassette Handler (C:)

The Cassette device is a read or write device with a Handler that supports the following CIO functions:

OPEN
CLOSE
GET CHARACTERS
GET RECORD
PUT CHARACTERS
PUT RECORD
GET STATUS (null function)

The Cassette Handler can produce the following error statuses:

$80 -- [BREAK] key abort.
$84 -- Invalid AUX1 byte on OPEN.
$88 -- end-of-file.
$8A-90 -- SIO error set (see Appendix C).

The Cassette Handler is one of the resident handlers, and therefore has a set of device vectors starting at location E440.

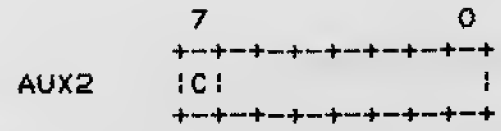
CIO Function Descriptions

The device-specific characteristics of the standard CIO functions are detailed below:

OPEN

The device name is C, and the Handler ignores any device number and filename specification, if included.

The Handler supports the following option:



Where: C = 1 indicates that the cassette is to be read/written without stop/start between records (continuous mode).

Opening the cassette for input generates a single audible tone, as a prompt for you to verify that the cassette player is set up for reading (power on; Serial Bus cable connected; tape cued to start of file; and PLAY button depressed). When the cassette is ready, you can press any keyboard key (except [BREAK]) to initiate tape reading.

Opening the cassette for output generates two closely spaced audible tones, as a prompt for you to verify that the cassette player is set up for writing (as above, plus RECORD button depressed). When the cassette is ready, you can press any keyboard key (except [BREAK]) to begin tape writing. There is no way for the computer to verify that the RECORD or PLAY button is depressed. It is possible for the file not to be written, with no immediate indication of this fact.

There is a potential problem with the cassette in that when the cassette is opened for writing, the motor keeps running until the first record (128 data bytes) is written. If 128 data bytes are written or the cassette is closed within about 30 seconds of the OPEN, and no other serial bus I/O is performed, then there is no problem. However, if those conditions are not met, some noise will be written to the tape prior to the first record and an error will occur when that tape file is read later. If lengthy delays are anticipated between the time the cassette file is opened and the time that the first cassette record (128 data bytes) is written, then a dummy record should be written as part of the file; typically 128 bytes of some innocuous data would be written, such as all zeros, all $FFs, or all blanks ($20).

The system sometimes emits whistling noises after cassette I/O has occurred. The sound can be eliminated by storing $03 to SKCTL [D20F], thus bring POKEY out of the two-tone (FSK) mode.

CLOSE

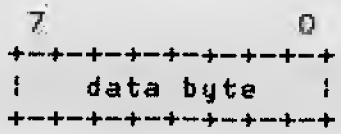
The CLOSE of a tape read stops the cassette motor.

The CLOSE of a tape write does the following:

Writes any remaining user data in the buffer to tape.
Writes an end-of-file record.
Stops the cassette motor.

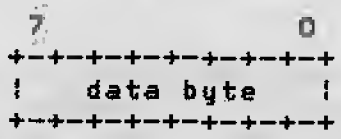
GET CHARACTERS and GET RECORD

The Handler returns data in the following format:



PUT CHARACTERS and PUT RECORD

The Handler accepts data in the following format:



The Handler attaches no significance to the data bytes written, a value of $9B (EOL) causes no special action.

GET STATUS

The Handler does no more than set the status to $01.

Theory of Operation

The Cassette Handler writes and reads all data in fixed-length records of the format shown below:

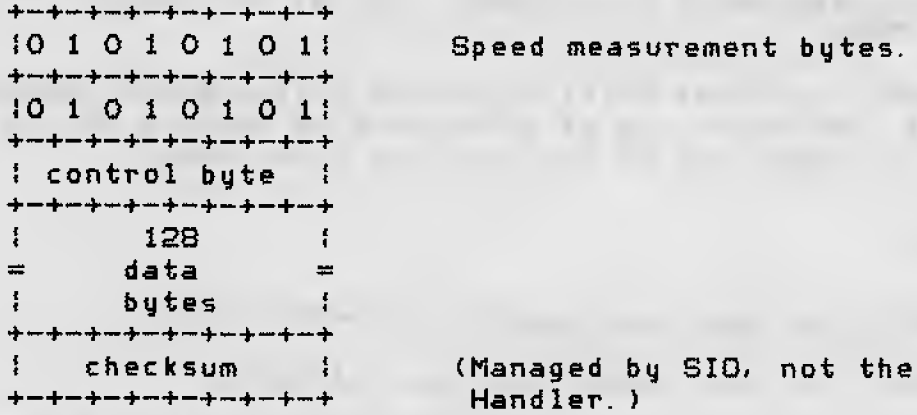


Figure 5-9 Cassette Handler Record Format

The control byte contains one of three values:

- $FC indicates the record is a full data record (128 bytes).

- $FA indicates the record is a partially full data record; you supplied fewer than 128 bytes to the record. This case can occur only in the record prior to the end-of-file. The number of user-supplied data bytes in the record is contained in the byte prior to the checksum.

- $FE indicates the record is an End-of file record; the data portion is all zeroes for an end-of-file record.

The SIO routine generates and checks the checksum. It is part of the tape record, but it is not contained in the Handler's record buffer CASBUF [03FD].

The processing of the speed-measurement bytes during cassette reading is discussed in Appendix L, D1-D7.

File Structure

The Cassette Handler writes a file to the cassette device with a file structure that is totally imposed by the Handler (soft format). A file consists of the following three elements:

- A 20-second leader of mark tone.

- Any number of data-record frames.

- An end-of-file frame.

The cassette-data record frames are formatted as shown below:

```
frame = pre-record write tone (PRWT),
        + data record,
        + post record gap (PRG)
```

The nondata portions of a frame have characteristics that are dependent upon the write OPEN mode, i.e. continuous or start/stop.

```
Stop/start PRWT = 3 seconds of mark tone.
Continuous PRWT = .25 second of mark tone.

Stop/start PRG = up to 1 second of unknown tones.
Continuous PRG = from 0 to n seconds of unknown tones, where
                 n is dependent upon your program timing.
```

The inter-record gap (IRG) between any two records consists of the PRG of the first record followed by the PRWT of the second record.

Printer Handler (P:)

The Printer device is a write-only device with a Handler that supports the following CIO functions:

OPEN
CLOSE
PUT CHARACTERS
PUT RECORD
GET STATUS

The Printer Handler can produce the following error statuses:

$8A-90 -- SIO error set (see Appendix C).

The Printer Handler is one of the resident handlers, and therefore has a set of device vectors starting at location E430.

CIO Function Descriptions

The device-specific characteristics of the standard CIO functions are detailed below:

OPEN

The device name is P. The Handler ignores any device number and filename specification, if included.

CLOSE

The Handler writes any data remaining in its buffer to the printer device, with trailing blanks to fill out the line.

PUT CHARACTERS and PUT RECORD

The Handler accepts print data in the following format:

The only ATASCII control code of any significance to the Handler is the EOL character. The printer device ignores bit 7 of every data byte and prints a sub set of the remaining 128 codes. (see Appendix G for the printer character set).

The Handler supports the following print option:

```
                7               0
               +-+-+-+-+-+-+-+-+
      AUX2     |  print mode   |
               +-+-+-+-+-+-+-+-+
```

Where: $4E (N) selects normal printing (40 characters per line).
$53 (S) selects sideways printing (29 characters per line).
$57 (W) selects wide printing (not supported by printer device).

Any other value (including 00) is treated as a normal (N) print select, without producing an error status.

GET STATUS

The Handler obtains a 4-byte status from the printer controller and puts it in system location DVSTAT [02EA]. The format of the status bytes is shown below:

```
+-+-+-+-+-+-+-+-+
| command stat. |     DVSTAT + 0
+-+-+-+-+-+-+-+-+
| AUX2 of prev. |            + 1
+-+-+-+-+-+-+-+-+
|    timeout    |            + 2
+-+-+-+-+-+-+-+-+
|   (unused)    |            + 3
+-+-+-+-+-+-+-+-+
```

The command status contains the following status bits and condition indications:

bit 0: an invalid command frame was received.
bit 1: an invalid data frame was received.
bit 7: an intelligent controller (normally = 0).

The next byte contains the AUX2 value from the previous operation.

The timeout byte contains a controller provided maximum timeout value (in seconds).

Theory of Operation

The ATARI 820[TM] 40-Column Printer is a line-at-a-time printer rather than a character-at-a-time printer, so your data must be buffered by the Handler and sent to the device in records corresponding to one print line (40 characters for normal, 29 characters for sideways).

The printer device does not attach any significance to the EOL character, so the Handler does the appropriate blank fill whenever it sees an EOL.

Disk File Manager (D:)

The OS supports four unique File Management Subsystems at the time of this writing. Version IA is the original version. Version IB is a slightly modified version of IA and is the one described in this document. Most of this discussion applies as well to Version II, that handles a double-density diskette (720 256-byte sectors) in addition to the single-density diskette (720 128-byte sectors). Version III has all new file/directory/map structures and can possibly contain changes to your interface as well.

The File Management Subsystem includes a disk-bootable (RAM-resident) Disk File Manager (DFM) that maintains a collection of named files on diskettes. Up to 4 disk drives (D1: through D4:) can be accessed, and up to 64 files per diskette can be accessed. The system diskettes supplied by ATARI allow a single disk drive (D1) and up to 3 OPEN files, but you can alter these numbers as described later in this section.

The Disk File Manager supports the following CIO functions:

OPEN FILE
OPEN DIRECTORY
CLOSE
GET CHARACTERS
GET RECORD
PUT CHARACTERS
PUT RECORD
GET STATUS

NOTE
POINT
LOCK
UNLOCK
DELETE
RENAME
FORMAT

The Disk File Manager can produce the following error statuses:

$03 -- Last data from file (EOF on next read).
$88 -- end-of-file.
$8A-90 -- SIO error set (see Appendix C).
$A0 -- Drive number specification error.
$A1 -- No sector buffer available (too many open files).
$A2 -- Disk full.
$A3 -- Fatal I/O error in directory or bitmap.
$A4 -- Internal file # mismatch (structural problem).
$A5 -- File name specification error.
$A6 -- Point information in error.
$A7 -- File locked to this operation.
$A8 -- Special command invalid.
$A9 -- Directory full (64 files).
$AA -- File not found.
$AB -- Point invalid (file not OPENed for update).

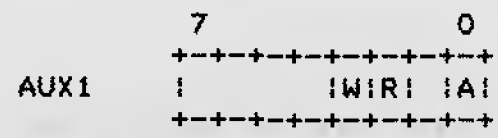
CIO Function Descriptions

The device-specific characteristics of the standard CIO functions are detailed below:

OPEN FILE

The device name is D. Up to four disk drives can be accessed (D1 through D4). The disk filename can be from 1 to 8 characters in length with an optional 1- to 3-character extension.

The OPEN FILE command supports the following options:



Where: W and R are the direction bits.
WR = 00 is invalid
01 indicates OPEN for read only.
10 indicates OPEN for write only.
11 indicates OPEN for read/write (update).

A = 1 indicates appended output when W = 1.

You may use these following valid AUX1 options:

OPEN Input (AUX1 = $04)

The indicated file is opened for input. Any wild-card characters are used to search for the first match. If the file is not found, an error status is returned, and no file will be opened.

OPEN Output (AUX1 = $08)

The indicated file is opened for output starting with the first byte of the file, if the file is not locked. Any wild-card characters are used to search for the first match. If the file already exists, the existing file will be deleted before opening the named file as a new file. If the file does not already exist, it will be created.

A file opened for output will not appear in the directory until it has been closed. If an output file is not properly closed, some or all of the sectors that were acquired for it can be lost until the disk is reformatted.

A file that is opened for output can not be opened concurrently for any other access.

OPEN Append (AUX1 = $09)

The indicated file is opened for output starting with the byte after the last byte of the existing file (that must already exist), if the file is not locked. Any wild-card characters are used to search for the first match.

If a file opened for append is not properly closed, the appended data will be lost. The existing file will remain unmodified and some or all of the sectors that were acquired for the appended portion can be lost until the diskette is reformatted.

OPEN Update (AUX1 = $0C)

The indicated file (that must already exist) will be opened for update provided it is not locked. Any wild-card characters are used to search for the first match.

The GET, PUT, NOTE and POINT operations are all valid, and can be intermixed as desired.

If a file opened for update is not properly closed, a sector's worth of information can be lost to the file. A file opened for update can not be extended.

Device/Filename Specification

The Handler expects to find a device/filename specification of the following form:

D[<number>]:<filename><EOL>

where:

<number> ::= 1|2|3|4
<filename> ::= [<primary>][.[<extension>]]<terminator>
<primary> ::= an uppercase alpha character followed by 0 to 7 alphanumeric characters. If the primary name is less than 8 characters, it will be padded with blanks; if it is greater than 8 characters, the extra characters will be ignored.

<extension> ::= Zero to 3 alphanumeric characters. If the extension name is missing or less than 3 characters, it will be padded with blanks; if it is greater than 3 characters, the extra characters will be ignored.

<terminator> ::= <EOL>|<blank>

Figure 5-10 Device/Filename Syntax

The following are all valid device/filenames for the diskette:

D1:GAME.SRC
D:MANUAL6
D:.WHY
D3:FILE.
D4:BRIDGE.002

Filename Wildcarding

The filename specification can be further generalized to include the use of the "wild-card" characters * and ?. These wildcard characters allow portions of the primary and/or extension to be abbreviated as follows:

The ? character in the specification allows any filename character at that position to produce a "match." For example, WH? will match files named WHO, WHY, WH4, etc., but not a file named WHAT.

The * character causes the remainder of the primary or extension field in that it is used to be effectively padded with ? characters. For example, WH* will match WHO, WHEN, WHATEVER, etc.

Some valid uses of wild-card specifications are shown below:

*.SRC	Files having an extension of SRC.
BASIC.*	Files named BASIC with any extension.
.	All files.
H*.?	Files beginning with H and having a 0 or 1 character extension.

If wildcarding is used with an OPEN FILE command, the first file found (if any) that meets the specification will be the one (and only one) opened.

OPEN DIRECTORY

The OPEN DIRECTORY command allows you read directory information for the selected filename(s), using normal GET CHARACTERS or GET RECORD commands. The information read will be formatted as ATASCII records, suitable for printing, as shown below. Wildcarding can be used to obtain information for multiple files or the entire diskette.

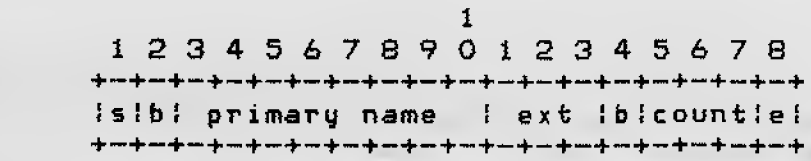
The OPEN DIRECTORY command uses the same CIO parameters as a standard OPEN FILE command:

COMMAND BYTE = $03

BUFFER ADDRESS = pointer to device/filename specification.

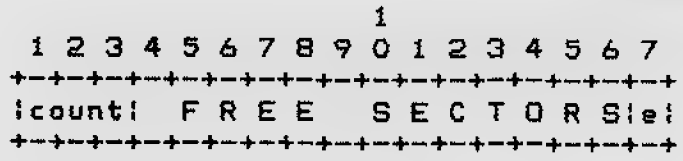
AUX1 = $06

After the directory is opened, a record will be returned to the caller for each file that matches the OPEN specification. The record, that contains only ATASCII characters, is formatted as shown below:



Where: s = * or ' ', with * indicating file locked.
b = blank.
primary name = left-justified name with blank fill.
ext = left-justified extension with blank fill.
b = blank.
count = number of sectors comprising the file.
e = EOL ($9B).

After the last filename match record is returned, an additional record is returned. This record indicates the number of unused sectors available on the diskette. The format for this record is shown below:



Where: count = the number of unused sectors on the diskette.
e = EOL ($9B).

The EOF statuses ($03 and $88) are returned as in a normal data file when the last directory record is read.

The opening of another diskette file while the directory read is open will cause subsequent directory reads to malfunction, so care must be taken to avoid this situation.

CLOSE

Upon closing a file read, the Handler releases all internal resources being used to support that file.

Upon closing a file write, the Handler:

- writes any residual data from its file buffer for that file to the diskette.
- updates the directory and allocation map for the associated diskette.
- releases all internal resources being utilized to support that file

GET CHARACTERS and GET RECORD

Characters are read from the diskette and passed to CIO as a raw data stream. None of the ATASCII control characters have any special significance. A status of $88 is returned if an attempt is made to read past the last byte of a file.

PUT CHARACTERS and PUT RECORD

Characters are obtained from CIO and written to the diskette as a raw data stream. None of the ATASCII control characters have any special significance.

GET STATUS

The indicated file is checked and one of the following status byte values is returned in ICSTA and register Y:

$01 -- File found and unlocked.
$A7 -- File locked.
$AA -- File not found.

Special CIO Functions

The DFM supports a number of SPECIAL commands, that are device specific. These are explained in the paragraphs that follow:

NOTE (COMMAND BYTE = $25)

This command returns to the caller the exact diskette location of the next byte to be read or written, in the variables shown below:

ICAX3 = LSB of the diskette sector number.
ICAX4 = MSB of the diskette sector number.
ICAX5 = relative sector displacement to byte (0-124).

POINT (COMMAND BYTE = $26)

This command allows you to specify the exact diskette location of the next byte to be read or written. In order to use this commmand, the file must have been opened with the "update" option.

ICAX3 = LSB of the diskette sector number.
ICAX4 = MSB of the diskette sector number.
ICAX5 = relative sector displacement to byte (0-124).

LOCK

This command allows you to prevent write access to any number of named files. Locked files can not be deleted, renamed, nor opened for output unless they are first unlocked. Locking a file that is already locked is a valid operation. The Handler expects a device/filename specification; then all occurrences of the filename specified will be locked, using the wild-card rules.

You set up these following IOCB parameters prior to calling CIO:

COMMAND BYTE = $23

BUFFER ADDRESS = pointer to device/filename specification.

After a LOCK operation, the following IOCB parameter will have been altered:

STATUS = result of LOCK operation; see Appendix B for a list of possible status codes.

UNLOCK

This command allows you to remove the lock status of any number of named files. Unlocking a file that is not locked is a valid operation. The Handler expects a device/filename specification; then all occurrences of the filename specified will be unlocked, using the wild-card rules.

You set up these following IOCB parameters prior to calling CIO:

COMMAND BYTE = $24

BUFFER ADDRESS = pointer to device/filename specification.

After an UNLOCK operation, the following IOCB parameter will have been altered:

STATUS = result of UNLOCK operation; see Appendix B for a list of possible status codes.

DELETE

This command allows you to delete any number of unlocked named files from the directory of the selected diskette and to deallocate the diskette space used by the files involved. The Handler expects a device/filename specification; then all occurences of the filename specified will be deleted, using the wild-card rules.

You set up these following IOCB parameters prior to calling CIO:

COMMAND BYTE = $21

BUFFER ADDRESS = pointer to device/filename specification.

After a DELETE operation, the following IOCB parameter will have been altered:

STATUS = result of DELETE operation; see Appendix B for a list of possible status codes.

RENAME

This command allows you to change the filenames of any number of unlocked files on a single diskette. The Handler expects to find a device/filename specification that follows:

<device spec>:<filename spec>,<filename spec><EOL>

All occurrences of the first filename will be replaced with the second filename, using the wild-card rules. No protection is provided against forming duplicate names. Once formed, duplicate names cannot be separately renamed or deleted; however, an OPEN FILE command will always select the first file found that matches the filename specification, so that file will always be accessible. The RENAME command does not alter the content of the files involved, merely the name in the directory.

Examples of some valid RENAME name specifications are shown below:

D1:*.SRC,*.TXT
D:TEMP,FDATA
D2:F*,F*.OLD

You set up these following IOCB parameters prior to calling CIO:

COMMAND BYTE = $20

BUFFER ADDRESS = pointer to device/filename specification.

After a RENAME operation, the following IOCB parameter will have been altered:

STATUS = result of RENAME operation; see Appendix B for a list of possible status codes.

FORMAT

Soft-sector diskettes must be formatted before they can store data. The FORMAT command allows you to physically format a diskette. The physical formatting process writes a new copy of every sector on the soft-sectored diskette, with the data portion of each sector containing all zeros. The FORMAT process creates an "empty" non system diskette. When the formatting process is complete, the FMS creates an initial Volume Table of Contents (VTOC) and an initial File Directory. The boot sector (#1) is permanently reserved as part of this process.

You set up these following IOCB parameters prior to calling CIO:

COMMAND BYTE = $FE

BUFFER ADDRESS = pointer to device specification.

After a FORMAT operation, the following IOCB parameter will have been altered:

STATUS = result of FORMAT operation; see Appendix B for a list of possible status codes.

To create a system diskette, a copy of the boot file must then be written to sectors #2-n. This is accomplished by writing the file named DOS.SYS. This is a name that is recognized by the FMS even though it is not in the directory initially.

Theory of Operation

The resident OS initiates the disk-boot process (see Section 10). The OS reads diskette sector #1 to memory and then transfers control to the "boot continuation address" (boot address + 6). The boot-continuation program contained in sector #1 then continues to load the remainder of the File Management Subsystem to memory using additional information contained in sector #1. The File Management Subsystem loaded, will contain a Disk File Manager ,and optionally, a Disk Utilities (DOS) package.

When the boot process is complete, the Disk File Manager will allocate additional RAM for the creation of sector buffers. Sector buffers are allocated based upon information in the boot record as shown below:

Byte 9 = maximum number of open files; one buffer per (the maximum value is 8).

Byte 10 = drive select bits; one buffer per (1-4 only).

The Disk File Manager will then insert the name D and the Handler vector table address in the device table.

NOTE: There is a discrepancy between the Disk File Manager's numbering of diskette sectors (0-719) and the disk controller's numbering of diskette sectors (1-720); as a result, only sectors 1- 719 are used by the Disk File Manager.

The Disk File Manager uses the Disk Handler to perform all diskette reads and writes; the DFM's function is to support and maintain the directory/file/bitmap structures as described in the following pages:

FMS Diskette Utilization

The map below shows the diskette sector utilization for a standard 720 sector diskette.

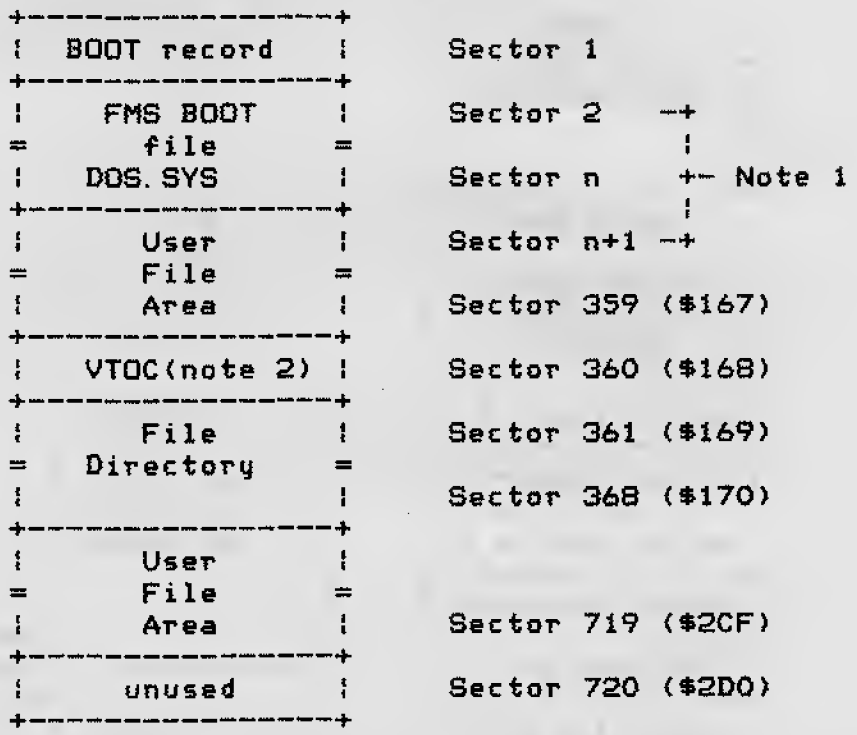


Figure 5-11 File Management Subsystem Diskette Sector Utilization Map

NOTE 1 - If the diskette is not a system diskette, then your File Area starts at sector 2 and no space is reserved for the FMS BOOT file. However, "DOS" (DOS.SYS and DUP.SYS) may still be written to a diskette that has already used sectors "2-N."

NOTE 2 -- VTOC stands for Volume Table of Contents.

FMS Boot Record Format

The FMS BOOT record (sector #1) is a special case of diskette-booted software (see Section 10). The format for the FMS BOOT record is shown below:

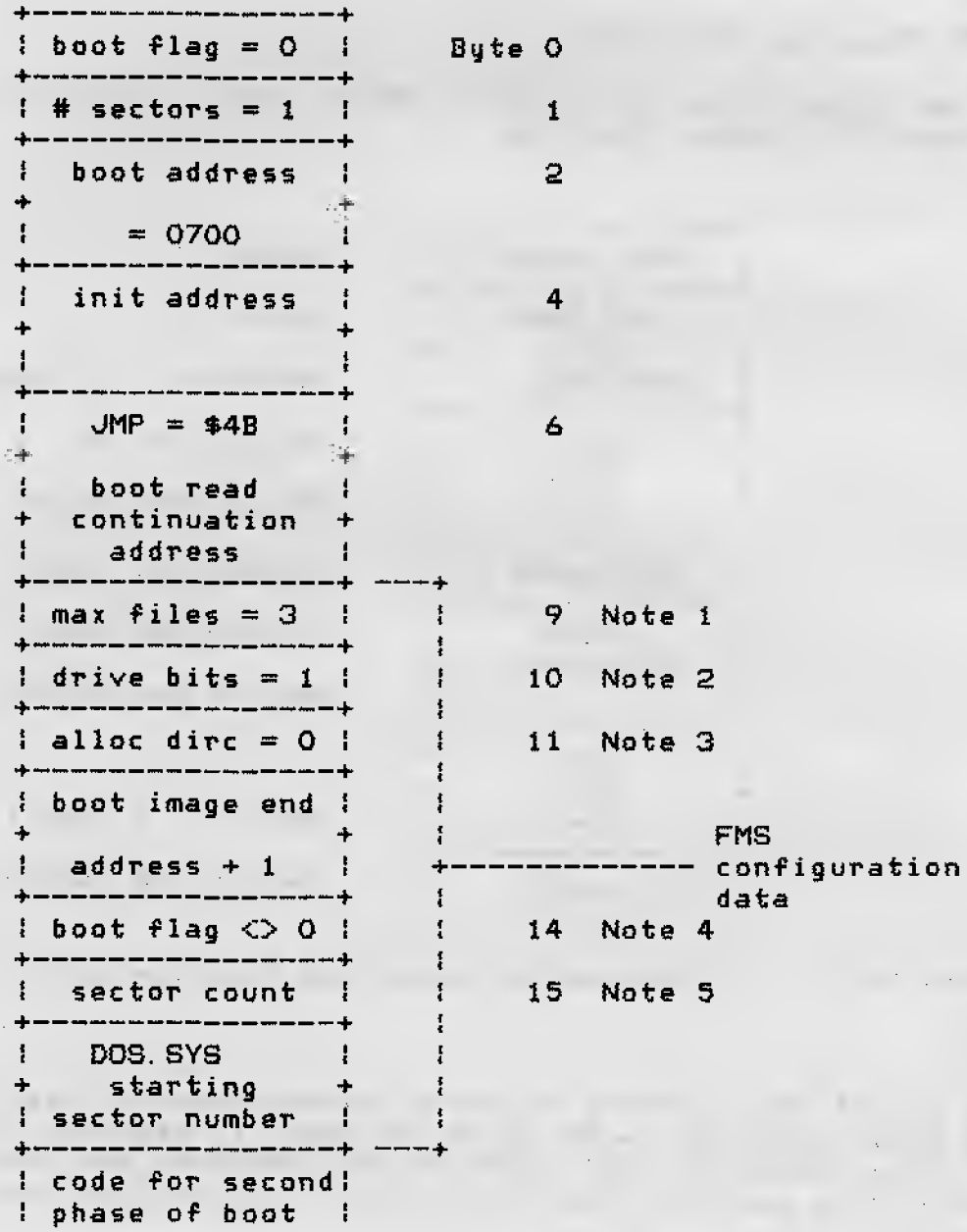
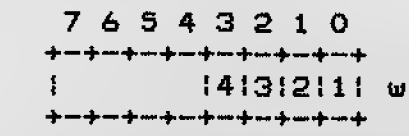


Figure 5-12 File Management Subsystem Boot Record Format

NOTE 1 - Byte 9 specifies the maximum number of concurrently open files to be supported. This value can range from 1 to 8.

NOTE 2 - Byte 10 specifies the specific disk drive numbers to be supported using a bit encoding scheme as shown below:



NOTE 3 - Byte 11 specifies the buffer allocation direction, this byte should equal 0.

NOTE 4 - Byte 14 must be nonzero for the second phase of the boot process to initiate. This flag indicates that the file DOS.SYS has been written to the diskette.

NOTE 5 - This byte is assigned as being the sector count for the DOS.SYS file. It is actually an unused byte.

Boot Process Memory Map

The diagram below shows how the boot sector (part of file DOS.SYS) and following sectors are loaded to memory as part of the boot process.

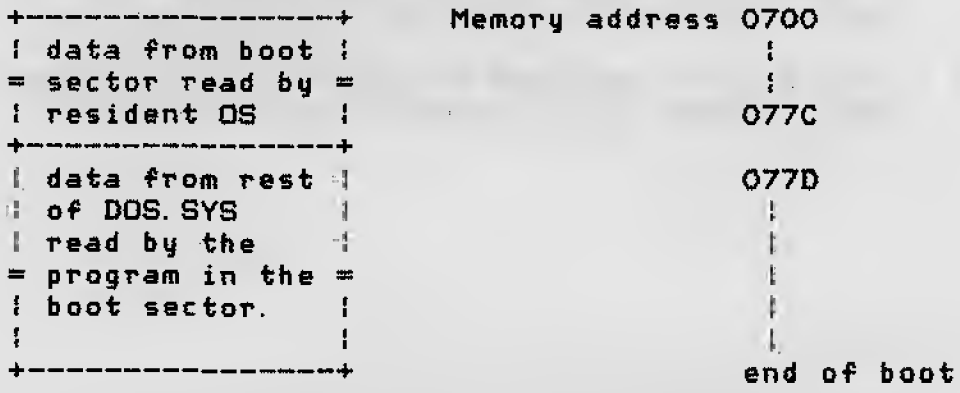


Figure 5-13 File Management Subsystem Boot Process Memory Map

Volume Table of Contents

The format for the FMS volume table of contents (VTOC, sector 360) is shown in the diagram below:

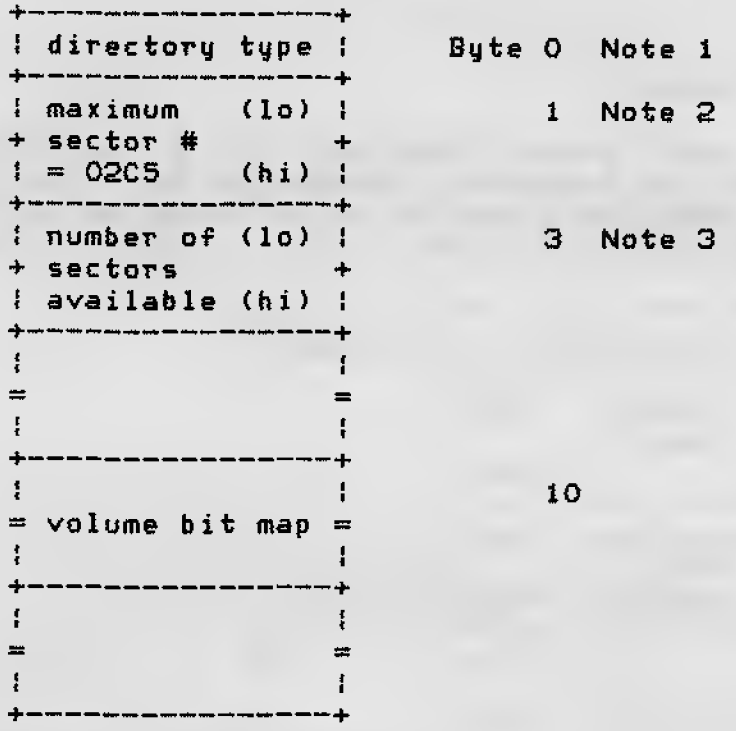


Figure 5-14 File Management Subsystem Volume Table of Contents

The volume bit map organization location follows:

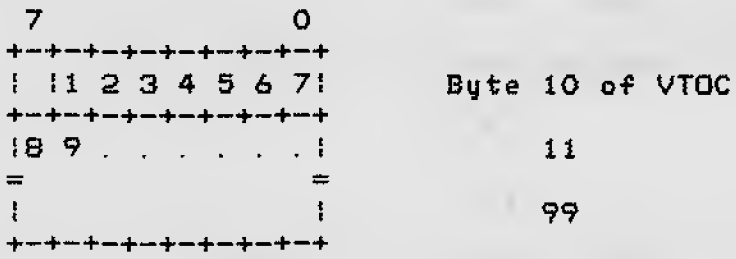


Figure 5-15 File Management Subsystem Volume Bit Map

At each map bit position, a 0 indicates the corresponding sector is in use and a 1 indicates that the sector is available.

NOTE 1 - The directory type byte must equal 0.

NOTE 2 - The maximum sector number is not used because it is incorrectly set to 709 decimal. The true maximum sector number is actually 719 for the DFM.

NOTE 3 - The number of sectors available is initially set to 709 after a diskette is freshly formatted; this number is adjusted as files are created and deleted to show the number of sectors available. The sectors that are initially reserved are 1 and 360-368.

File Directory Format

The FMS reserves eight sectors (361-368) for a file directory. Each sector containing directory information for up to eight files, thus providing for a maximum of 64 files for any volume. The format of a single 16-byte file entry is shown below:

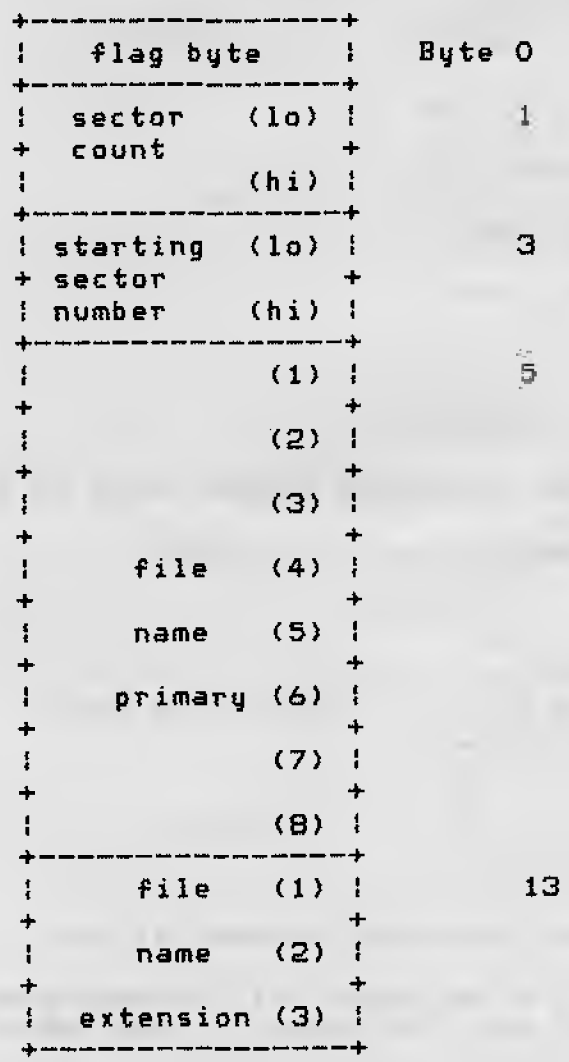


Figure 5-16 File Directory Format

Where the flag byte has the following bits assigned:

bit 7 = 1 if the file has been deleted.
bit 6 = 1 if the file is in use.
bit 5 = 1 if the file is locked.
bit 0 = 1 if OPEN output.

The flag byte can take on the following values:

$00 = entry not yet used (no file).
$40 = entry in use (normal CLOSEd file).
$41 = entry in use (OPEN output file).
$60 = entry in use (locked file).
$80 = entry available (prior file deleted).

Sector count is the number of sectors comprising the file.

FMS File Sector Format

The format of a sector in your data file is shown below:

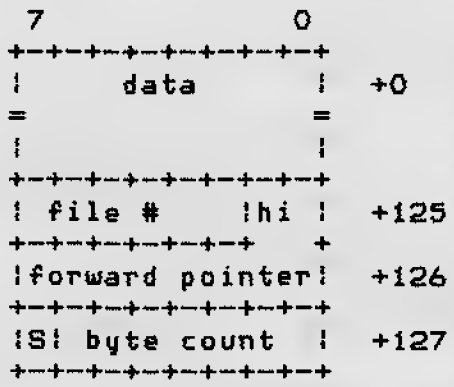


Figure 5-17 File Management Subsystem File Sector Format

The FMS uses the file # to verify file integrity. The file # is a redundant piece of information. The file number field contains the value of the directory position of that file. If a mismatch occurs between the file's directory position, and the file number as contained in each sector, then the DFM will generate the error $A4.

The forward pointer field contains the 10-bit value for the diskette sector number of the next sector of the file. The pointer equals zero for the last sector of a file.

The S bit indicates whether or not the sector is a "short sector" (a sector containing fewer than 125 data bytes). S is equal to 1 when the sector is short.

The byte-count field contains the number of data bytes in the sector.

Non-CIO I/O

Some portions of the I/O subsystem are accessed independently of the Central I/O Utility (CIO); this section discusses those areas.

Resident Device Handler Vectors

All of the OS ROM resident device handlers can be accessed via sets of vectors that are part of the OS ROM. These vectors increase the speed of I/O operations that utilize fixed device assignments, such as output to the Display Handler. For each resident Handler there is a set of vectors ordered as shown below:

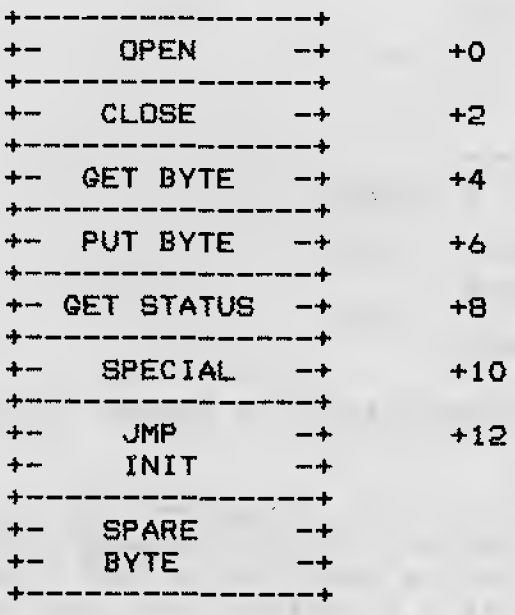


Figure 5-18 Resident Device Handler Vectors

See Section 9 for a detailed description of the data interface for each of these Handler entry points.

Each of the vectors contains the address (lo,hi) of the Handler entry point minus 1. A technique similar to the one shown below is required to access the desired routines:

```
VTBASE=$E400                    ; BASE OF VECTOR TABLE.

        .
        LDX     #xx             ; OFFSET TO DESIRED ROUTINE.
        LDA     data
        JSR     GOVEC           ; SEND DATA TO ROUTINE.
        .
        LDX     #yy             ; OFFSET TO DIFFERENT ROUTINE.
        JSR     GOVEC           ; GET DATA FROM ROUTINE.
        STA     data
        .

GOVEC   TAY                     ; SAVE REGISTER A.
        LDA     VTBASE+1,X      ; ADDRESS MSB TO STACK.
        PHA
        LDA     VTBASE,X        ; ADDRESS LSB TO STACK.
        PHA
        TYA                     ; RESTORE REGISTER A.
        RTS                     ; JUMP TO ROUTINE.
```

The JMP INIT slot in each set of vectors jumps to the Handler initialization entry (not minus 1).

The base address of the vector set for each of the resident handlers is shown below:

Screen Editor (E:)	E400.
Display Handler (S:)	E410.
Keyboard Handler (K:)	E420.
Printer Handler (P:)	E430.
Cassette Handler (C:)	E440.

The resident diskette Handler is not CIO-compatible, so its interface does not use a vector set.

Resident Diskette Handler

The resident Diskette Handler (not to be confused with the Disk File Manager) is responsible for all physical accesses to the diskette. The unit of data transfer for this Handler is a single diskette sector containing 128 data bytes.

Communication between you and the Diskette Handler is effected using the system's Device Control Block (DCB), that is also used for Handler/SIO communication (see Section 9). The DCB is 12 bytes long. Some bytes are user-alterable and some are for use by the Diskette Handler and/or the Serial I/O Utility (SIO). You supply the required DCB parameters and then do a JSR DSKINV [E453].

Each of the DCB bytes will now be described, and the system-equate file name for each will be given.

SERIAL BUS ID -- DDEVIC [0300]

The Diskette Handler sets up this byte to contain the Serial Bus ID for the drive to be accessed. It is not user-alterable.

DEVICE NUMBER -- DUNIT [0301]

You set up this byte to contain the disk drive number to be accessed (1 - 4).

COMMAND BYTE -- DCOMND [0302]

You set up this byte to contain the disk device command to be performed.

STATUS BYTE -- DSTATS [0303]

This byte contains the status of the command upon return to the caller. See Appendix C for a list of the possible status codes.

BUFFER ADDRESS -- DBUFLO [0304] and DBUFHI [0305]

This 2-byte pointer contains the address of the source or destination of the diskette sector data. You need not supply an address for the disk status command. The Disk Handler will obtain the status and insert the address of the status buffer into this field.

DISK TIMEOUT VALUE -- DTIMLO [0306]

The Handler supplies this timeout value (in whole seconds) for use by SIO.

BYTE COUNT -- DBYTLO [0308] and DBYTHI [0309]

This 2-byte counter indicates the number of bytes transferred to or from the disk as a result of the most recent command, and is set up by the Handler.

SECTOR NUMBER -- DAUX1 [030A] and DAUX2 [030B]

This 2-byte number specifies the diskette sector number (1 - 720) to read or write. DAUX1 contains the least significant byte, and

DAUX2 contains the most significant byte.

Diskette Handler Commands

There are five commands supported by the Diskette Handler:

GET SECTOR (PUT SECTOR --*** not supported by current handler ***)
PUT SECTOR WITH VERIFY
STATUS REQUEST
FORMAT DISK

GET SECTOR (Command byte = $52)

The Handler reads the specified sector to your buffer and returns the operation status. You set the following DCB parameters prior to calling the Diskette Handler:

COMMAND BYTE = $52.

DEVICE NUMBER = disk drive number (1-4).

BUFFER ADDRESS = pointer to your 128-byte buffer.

SECTOR NUMBER = sector number to read.

Upon return from the sector, several of the other DCB parameters will have been altered. The STATUS BYTE will be the only parameter of interest to you, however.

PUT SECTOR (Command byte = $50)

*** Not supported by current Handler ***
(But can be accessed through SIO directly.)

The Handler writes the specified sector from your buffer and returns the operation status. You set the following DCB parameters prior to calling the Diskette Handler:

COMMAND BYTE = $50.

DEVICE NUMBER = disk drive number (1-4).

BUFFER ADDRESS = pointer to your 128 byte buffer.

SECTOR NUMBER = sector number to write.

Upon return from the operation, several of the other DCB parameters will have been altered. The STATUS BYTE will be the only one of interest you, however.

PUT SECTOR WITH VERIFY (Command Byte = $57)

The Handler writes the specified sector from your buffer and returns the operation status. This command differs from PUT SECTOR in that the diskette controller reads the sector data after writing to verify the write operation. Aside from the COMMAND BYTE value, the calling sequence is identical to PUT SECTOR.

STATUS REQUEST (Command byte = $53)

The Handler obtains a 4-byte status from the diskette controller and puts it in system location DVSTAT [02EA]. The operation status format is shown below:

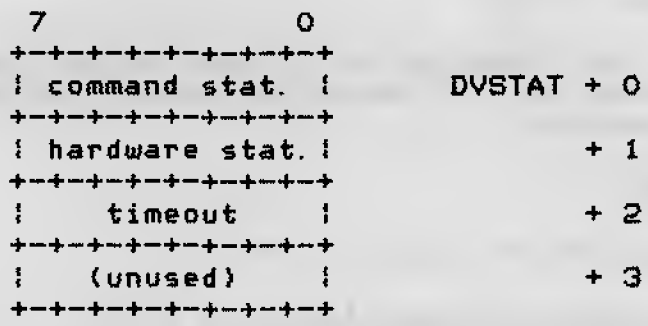


Figure 5-19. DVSTAT 40-Byte Operation Status Format

The command status contains the following status bits:

Bit 0 = 1 indicates an invalid command frame was received.
Bit 1 = 1 indicates an invalid data frame was received.
Bit 2 = 1 indicates that a PUT operation was unsuccessful.
Bit 3 = 1 indicates that the diskette is write protected.
Bit 4 = 1 indicates active/standby.

The hardware status byte contains the status register of the INS1771-1 Floppy Diskette Controller chip used in the diskette controller. See the documentation for that chip to obtain information relating to the meaning of each bit in the byte.

The timeout byte contains a controller-provided maximum timeout value (in seconds) to be used by the Handler.

You set the following DCB parameters prior to calling the Diskette Handler:

COMMAND BYTE = $53.

DEVICE NUMBER = disk drive number (1-4).

Upon return from the operation, several of the other DCB parameters will have been altered. The STATUS BYTE will be the only one of

interest to you, however.

FORMAT DISK (Command Byte = $21)

The Handler commands the diskette controller to format the entire diskette and then to verify it. All bad sector numbers (up to a maximum of 63) are returned and put in the supplied buffer, followed by two bytes of all 1's ($FFFF). You set up the following DCB parameters prior to calling the Diskette Handler:

COMMAND BYTE = $21.

DEVICE NUMBER = disk drive number (1-4).

BUFFER ADDRESS = pointer to your 128-byte buffer.

Upon return, you might be interested in the following DCB parameters:

STATUS BYTE = status of operation.

BYTE COUNT = number of bytes of bad sector information in your buffer, not including the $FFFF terminator. If there are no bad sectors, the count will equal zero.

Serial Bus I/O

Input/Output to devices other than the keyboard, the screen, and the ATARI Computer controller port devices, must utilize the Serial I/O bus. This bus contains data, control, and clock lines to be used to allow the computer to communicate with external devices on this "daisychained" bus. Every device on the bus has a unique identifier and will respond only when directly addressed.

The resident system provides a Serial I/O Utility (SIO), that provides a standardized high-level program interface to the bus. SIO is utilized by the resident Diskette, Printer, and Cassette handlers, and is intended to be used by nonresident handlers (see Section 9), or by applications, as well. For a detailed description of the program/SIO interface and for a detailed bus specification refer to Section 9.